TÉCNICO LISBOA

Cryptocurrency price direction prediction through ensembles of machine learning algorithms allied with percentage resampling

Pedro Fernandes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Rui Fuentecilla Maia Ferreira Neves

Examination Committee

Chairperson: Prof. Mário Jorge Costa Gaspar da Silva
Supervisor: Prof. Rui Fuentecilla Maia Ferreira Neves
Member of the Committee: Prof. Cláudia Martins Antunes

October 2022

Abstract

This work presents a system with the aim of generating a profitable bitcoin trading strategy, based on machine learning models trained on historical data. Before defining our algorithms' target variable, a resampling was performed to group sequential data points that lead to absolute percentage price movements of around 4%. Multiple technical and time-based features were generated and the problem was then approached as a binary task. Four different machine learning models were trained - Logistic Regression, Support Vector Machine, Random Forest, and XGBoost. Different trading strategies were then generated, either directly from the individual models' predictions or from different ways of combining them, and evaluated in a 9-month trading simulation period. The results proved that every single model beat the Buy and Hold strategy used as a baseline. Furthermore, the best single-model strategy (XGBoost) had a Return on Investment of 94.78%, and the best ensemble strategy achieved 119,76% for the same metric, over a period of 9 months. These results were obtained during a turbulent period, where bitcoin's price decreased by over 30%.

Keywords

Machine Learning; Logistic Regression; Support Vector Machine; Random Forest; XGBoost; Ensemble Voting; Bitcoin; Technical Analysis.

Resumo

Este trabalho apresenta um sistema com o objectivo de gerar uma estratégia lucrativa de negociação de bitcoin, baseada em modelos de aprendizagem automática treinados sobre dados históricos. Antes de definir a variável alvo dos nossos algoritmos, uma agregação foi realizada para agrupar pontos de dados sequenciais que levam a alterações percentuais absolutas de cerca de 4%. Foram geradas múltiplas características técnicas e baseadas em tempo, e o problema foi então abordado como uma tarefa binária. Foram treinados quatro algoritmos diferentes de aprendizagem de máquina - Regressão Logística, Máquina Vectorial de Apoio, Floresta Aleatória e XGBoost. Foram então geradas diferentes estratégias de compra e venda, quer directamente a partir das previsões dos modelos individuais ou de diferentes formas de os combinar, e avaliadas num período de simulação de mercado de 9 meses. Os resultados provaram que cada um dos modelos bateu a estratégia Buy and Hold utilizada como comparação base. Além disso, a melhor estratégia de modelo único (XGBoost) teve um retorno do investimento de 94,78%, e a melhor estratégia conjunta atingiu 119,76% para a mesma métrica, ao longo de um período de 9 meses. Estes resultados foram obtidos durante um período turbulento, em que o preço do bitcoin diminuiu mais de 30%.

Palavras Chave

Aprendizagem da Máquina; Regressão Logística; Máquina Vectorial de Apoio; Floresta Aleatória; XGBoost; Votação em Conjunto; Bitcoin; Análise Técnica.

Contents

List of Figures

List of Tables

Acronyms

RF	Random Forest
SVM	Support Vector Machine
RSI	Relative Strength Index
MACD	Moving average convergence Divergence
EMA	Exponential Moving Average
LSTM	Long Short Term Memory
KNN	K-Nearest Neighbors
ANN	Artificial Neural Network
ML	Machine Learning
ATR	Average True Range
V-ROC	Volume Rate of Change
GBM	Gradient Boosting Machine
RBF	Radial Basis Function
GTB	Gradient Tree Boosting
TIM	Time in Market
ROI	Return on Investment

1

Introduction

Contents

Launched in 2009 [1], Bitcoin's popularity and price have skyrocketed in recent years. Due to its frequent significant price movements, bitcoin's high-risk/high-reward profile quickly attracted the attention of individual investors [2] and, more recently, institutional investors [3].

Like bitcoin, there are hundreds of other cryptocurrencies being traded everyday through cryptocurrency exchanges like Binance [4]. Recently, the total cryptocurrency market capitalization (calculated by multiplying the price of the cryptocurrency with the number of coins in circulation) has spiked from 150 billion in March of 2020 to around 2.5 trillion in November of 2021 [5]. This surge in price has seen a proportional surge in interest, both from individual and institutional investors. Supported by its potential for extreme and fast profit, the cryptocurrency market has quickly become one of the largest unregulated markets in the world [6].

The cryptocurrency market is included in what is called a financial market - a system that provides buyers and sellers the means to trade financial instruments, including bonds, equities, the various international currencies, and derivatives. Due to this, many believe that it is possible to exploit the strategies utilized in other financial markets (e.g. stock or foreign exchange markets) like technical analysis and algorithmic trading. On the other hand, some experts believe that the market cannot be timed and, therefore, the best strategy would be to Buy and Hold. In this work, multiple algorithmic trading strategies were designed by machine learning models based on technical analysis, and compared to the Buy and Hold strategy.

1.1 Motivation

Due to its recency, the cryptocurrency market is still highly speculative which leads to an extremely volatile market. This means that price changes in the range of 5 to 10% in a single day are not out of the ordinary [10]. For that reason, the cryptocurrency market appears to be an interesting opportunity to apply machine learning methods using what is already known from the traditional stock market but also exploring problem-specific features.

1.2 Objectives

The main objective of this work is to build a machine learning model capable of accurately predicting the price movement of bitcoin and making a profit even when the general trading fees are accounted for.

We intend to improve on the state-of-the-art by applying, combining and improving on the findings of previous works that will be further analyzed in Section 2. We will study and compare the performance of various machine learning algorithms that proved effective in previous works, and also explore data preparation, feature engineering, and hyper-parameter tuning techniques.

In our work, the task of predicting the cryptocurrency's price movement will be approached as a binary classification problem with tabular data. To capitalize on the cryptomarket's high volatility and regular significant price changes, a resampling will be performed by grouping 1-min candles that lead to a specific price percentage change (around 4%), and these new candles will be our target labels. To feed our model, two categories of features will be used:

1. Traditional technical analysis
2. Time-based information

Regarding the first category, our models will operate with previously tested and well-known technical indicators such as the RSI and MACD indicators. As our data will be resampled based on percentage change, a need arises to maintain our models' notion of time, thus leading us to the second category - Time-based information.

1.3 Thesis Contribution

The main contributions from this work are:

1. The creation of a new percentage-based resampling technique, that better aligns the target variable with the objective of a system of this nature (to be profitable);
2. The introduction of an equation that allows for the labelling of positive and negative asymmetric candles that lead to break-even positions when they happen sequentially;
3. Training multiple models that overwhelmingly beat the Buy and Hold strategy, during the entire 284-day test period.

1.4 Document Structure

This document is structured as follows:
In Section 2, some background concepts are discussed and related literature is reviewed. In Section 3, the proposed solution is described and each of its components is thoroughly detailed. Section 4 goes over the results and three case studies, designed to analyze specific parameters and their optimal values. Section 5 concludes.

2

Related Work

Contents

The following Section is divided into two parts: first, a brief description of several Machine Learning and Financial Market concepts will be given. Afterward, related literature is reviewed and the state-of-the-art is analyzed.

2.1 Background

The following subsections will give a brief description of several Machine Learning and Financial Market concepts.

2.1.1 Cryptocurrency

Cryptocurrency is defined by Oxford Languages [7] as a ”digital currency in which transactions are verified and records maintained by a decentralized system using cryptography, rather than by a centralized authority”. Most cryptocurrencies are enabled by blockchain technology - a distributed ledger maintained by a decentralized network of computers.

The first blockchain-based cryptocurrency was launched in 2009 by the unknown entity ”Satoshi Nakamoto”. In his whitepaper entitled ”Bitcoin: A Peer-to-Peer Electronic Cash System” [1], Satoshi described the original plan, protocol, and vision for bitcoin which was supposed to facilitate online transactions and eliminate the need for a trusted third party, but, due to high transaction costs, slow transaction confirmations and its limited supply, bitcoin has moved away from Satoshi's vision of electronic cash and become closer to a highly volatile version of digital gold [8].

2.1.2 Cryptocurrency Market

As mentioned before, cryptocurrencies are traded in public exchanges like Binance and Coinbase. As of December 2021, there are 310 cryptocurrency exchanges and over 8000 cryptocurrencies listed on the CoinMarketcap website [9]. Unlike the stock market, cryptocurrency exchanges are open 24 hours a day, 7 days a week allowing for more trading opportunities, especially in the case of algorithmic trading. Another advantage of a 24/7 market is the property of continuity of signal: with traditional stocks, technical-analysis-based algorithms often struggle after signal interruptions, especially when important news drop during that time (for example earnings calls after trading hours). One disadvantage of the cryptocurrency market is the high transaction fees. In this work, Binance's standard fee of 0.1% will be considered.

2.1.3 Candlestick charts

Candlestick charts originated in Japan, in the 1700s and contain information that is used by multiple technical indicators. The basic idea of a candlestick chart is to illustrate information about the relation between two assets' values (usually one of the assets is currency), throughout time. The most common form of technical analysis comes from analyzing candlestick charts in an attempt to find patterns. Figure 2.1 shows a traditional candlestick chart with hourly candles of the pair Bitcoin / USD.

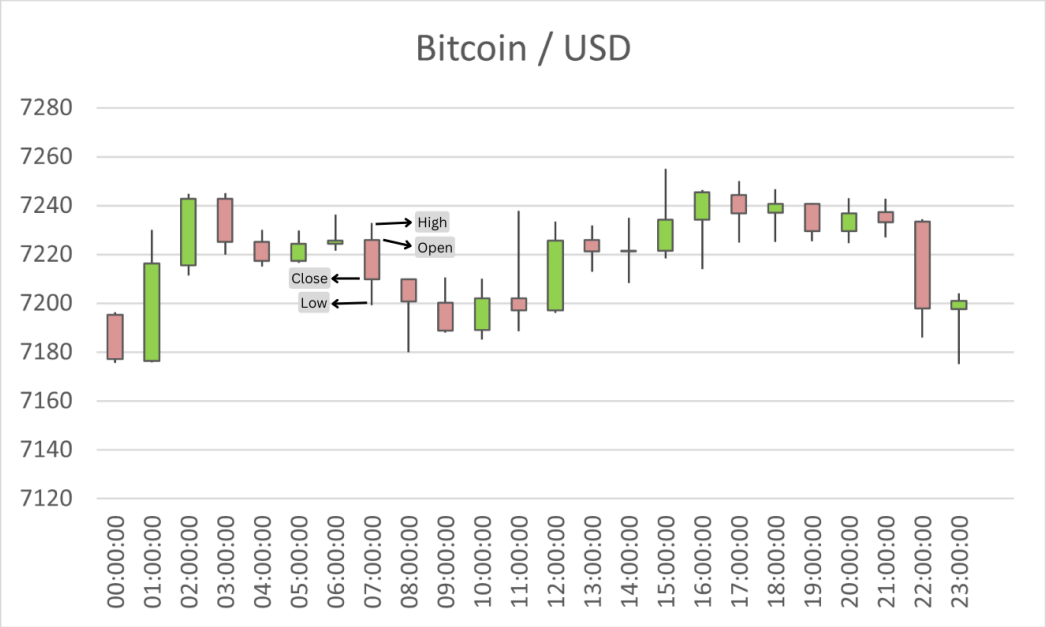

Figure 2.1: Candlestick chart with hourly candles of the pair Bitcoin / USD

The basic unit of a candlestick chart is a candle. In a traditional candlestick chart, every candle represents the same time length - it could be one minute, one day, or even one month. From observing each candle we can learn the highest, lowest, opening, and closing prices for that specific period. The total asset volume traded and the total number of trades can also usually be found in a separate graph and associated with each period.

In this work, the traditional candlestick chart will be resampled from time-based candles to percentage-change-based candles. This will be further explained in Section 3.3.1.

2.1.4 Machine Learning

Machine learning is a field of artificial intelligence that studies computer algorithms that use historical data to predict future outcomes. To achieve this, one must provide a great amount of data to the algorithm so that it can recognize patterns and apply them to current data to make predictions. Machine learning tasks can be split into two categories: classification and regression. In a classification problem, the algorithm must predict a label or a class. On the other hand, in a regression problem, the algorithm must predict a real quantity.

A time series is a sequential set of data points, measured typically over successive times [10], usually with equal intervals between them. By nature, financial time series are dynamic, non-linear, non-stationary, and noisy [11] making prediction tasks more challenging. Therefore, before feeding our machine learning algorithms, a great deal of data preparation must be done. We also made use of well-known financial technical indicators to remove the noise from the data, as these already perform operations on sliding windows, thus smoothing the data.

2.1.5 Hyper-parameters

Hyper-parameters are passed on to machine learning algorithms and define how they learn a task. For example, in a tree-based algorithm such as Random Forest (RF), one needs to define the number of decision-trees that the model will have, how deep these trees are, and many other hyper-parameters that help shape the final model.

In a problem like this, setting the right hyper-parameters for our algorithms can make the difference between our models being profitable or useless. Given this, a lot of effort was put into tuning these hyper-parameters through Bayesian optimization. This technique and all of the tuned hyper-parameters will be explained in detail in Section 3.3.1.

2.1.6 Model Testing

Generally speaking, there are two strategies to test a machine learning classification model: Holdout and K-Fold Cross Validation.

The Holdout strategy consists of separating our data-set into a train-set and a test-set. The split percentages are usually around 70% train-set and 30% test-set. The algorithm will then learn from the train-set and the resulting model will try to predict the class of each record that belongs to the test-set. The predictions made on unseen data are then evaluated through metrics such as Accuracy, Precision, and Recall.

On the other hand, K-Fold Cross Validation separates the data-set into K folds (usually K is between five and ten). The model will then train on K-1 folds, and use the remaining fold as the test-set.

K iterations are run and each fold is used as the test-set once. By saving the results after each iteration, and after all iterations are done, metrics like the average accuracy are taken into account to measure the model's performance.

Both of these strategies were used in this work, but in different steps: Holdout was used as the main testing strategy in this work because the test set had to be posterior to the train set. This is based on two reasons:

1. To prevent the model from learning future behavior. As this problem is based on time-series data, our model should not be allowed to learn from data that is posterior to the one it is making its predictions on. This is something that would not be possible in a real scenario;

2. So that a trading simulation could be performed. In a problem like this one, machine learning metrics alone do not tell us if our strategy is profitable. One must use the predictions to generate trade signals in a trading simulation.

K-Fold was used in the hyper-parameter tuning phase, due to the smaller amount of records and the fact that these results didn't have to be financially analyzed. As only the training set is passed on to this module, future data was not an issue here.

2.1.7 Data preparation

Data preparation is the process of cleaning and transforming raw data into relevant and usable features that we can feed to our machine learning models. It includes, but is not limited to, Variable Encoding, Data Scaling, Data Balancing, and Feature Engineering.

2.1.7.A Variable Encoding

Variables can be split into numeric or categorical depending on the information that they convey: Numerical variables can take any real number. On the other hand, categorical variables store one of a limited number of possible values. Categorical variables can be further divided into nominal and ordinal variables, with the difference being that, in the latter, there are categories that can be placed in distinct order or hierarchy, while in the former, no relationship between values is observed [12].

As some algorithms deal strictly with numeric values, it is important to translate categorical features into numeric ones.

For ordinal variables, it is important to maintain the hierarchy, therefore, most times integers are used to label the categories in accordance with the previous hierarchy. However, some ordinal variables are cyclical (e.g. day of the week). By labeling the day of the week from 1 to 7 (Monday = 1, Sunday = 7), our algorithms (especially distance-based ones) may lose sense of the fact that Monday is really

close to Sunday. One way of correctly encoding cyclical variables is to describe them through cyclical functions. First, one must normalize the variable values to the range $[0, 2\pi]$, afterward, simply compute the corresponding sine and cosine values for each value. Both functions are needed as both of them produce duplicate outputs for different inputs, but when combined, different inputs always generate unique pairs of values. An example of sine and cosine encoding is provided in Table 2.1. The resulting points are then plotted in Figure 2.2.

Day	Normalization	Cosine	Sine
Monday	(1 / 7) * 2 PI	0,623	0,782
Tuesday	(2 / 7) * 2 PI	-0,223	0,975
Wednesday	(3 / 7) * 2 PI	-0,901	0,434
Thursday	(4 / 7) * 2 PI	-0,901	-0,434
Friday	(5 / 7) * 2 PI	-0,223	-0,975
Saturday	(6 / 7) * 2 PI	0,623	-0,782
Sunday	(7 / 7) * 2 PI	1	0

Table 2.1: Days of the week encoding.

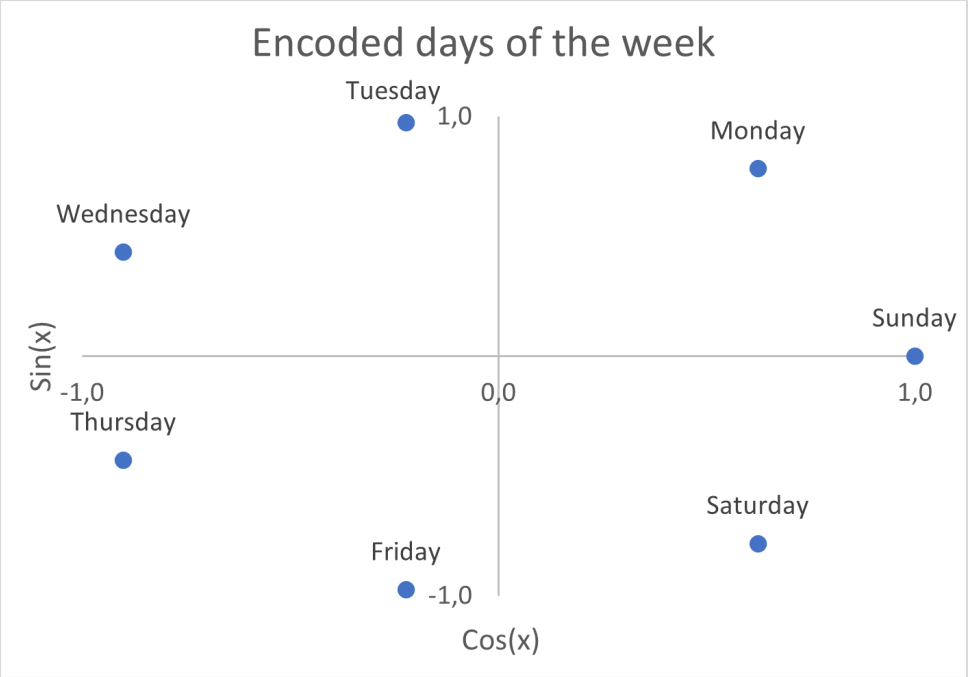

Figure 2.2: Encoded days of the week plotted in a scatter plot.

As for nominal variables, using an ordinal encoding might create an order where no such relationship may exist [13]. To avoid this, dummy encoding can be used. Dummy variable encoding works by transforming a categorical variable into a set of binary variables. An example of dummy encoding can be seen in Figure 2.3.

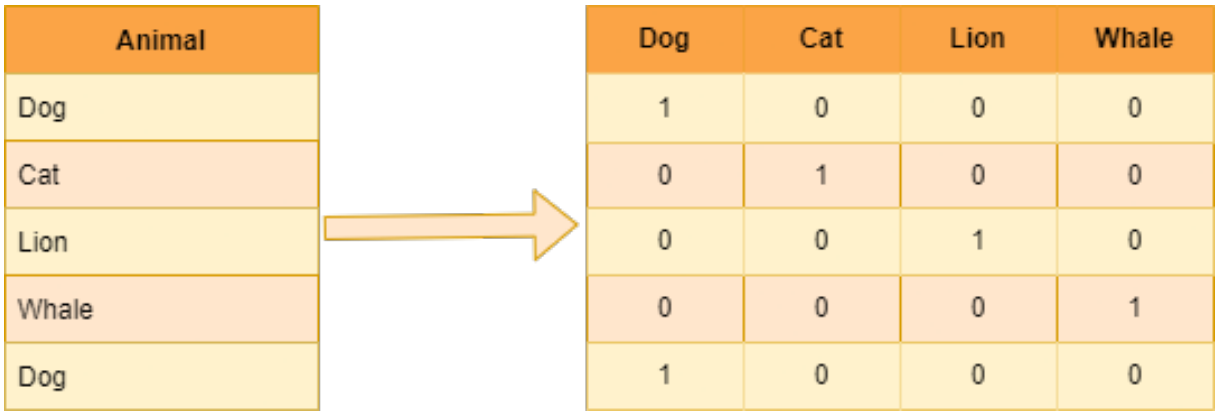

Figure 2.3: Nominal variable dummy encoding example

2.1.7.B Data Scaling

In the Data Scaling step, the features are transformed so that they all fit into the same scale. This could be, for example, the 0-1 scale or the 0-100 scale. This transformation prevents Gradient Descent Based algorithms (Linear and Logistic Regression, Neural Network) and Distance-based algorithms (K-Nearest Neighbors (KNN), K-means, Support Vector Machine (SVM)) from having geometrical biases towards features that have a greater range of values. Without scaling, and given the nature of these classifiers, these features would be given more importance over features that have a smaller range of values.

2.1.7.C Data Balancing

Most times, when working with real data sets, the Data Balancing step is necessary to avoid having an imbalanced class distribution (skewed class proportions), as this will cause the machine learning algorithm to be biased towards the majority class [38]. Given the challenging nature of this problem, it is expected that our models never get too far beyond the 55% accuracy mark. So it becomes especially important to keep the training set balanced as a small class imbalance might tip our models' predictions heavily to the majority class.

There are three possible strategies for balancing the training-set classes:

1. Over-sampling the minority class by creating synthetic records;
2. Under-sampling the majority class by removing records;
3. A mixture of both under- and over-sampling.

In this work, the chosen strategy was to under-sample the majority class, more specifically, a random sample of the majority class was selected, while the rest of the records belonging to this class were discarded.

2.1.7.D Feature engineering

Feature engineering is the process of using domain knowledge to extract and create relevant features to feed machine learning models. In this work, we will start by resampling our financial series using a novel method proposed by Borges et al. [14], which is discussed in the Related Work Section, and proceed to compute relevant technical and time-based features. A few technical-based indicators are explained below. As for the time-based features, they will be explained in Section 3.3, as they are specific to this work and, therefore, were not included in the background concepts.

2.1.8 Technical features

Technical indicators are used to find the best points of entry and exit when trading in financial markets by identifying patterns that have happened in the past in order to predict future price movements. This type of technical analysis has been a common practice for a long time now and was first introduced by Charles Dow in the late 1800s.

As technical indicators have become widely known and used, one can argue that their predictive power may be explained, in part, as self-fulfilling prophecies. Today, hundreds of technical indicators are available and, as such, it is important to understand them so we can apply the most relevant and effective ones regarding our specific problem. A brief explanation of a few technical indicators follows. A more complete overview can be found in [18]. There are four main categories of technical indicators:

1 **Trend indicators**

These indicators measure the direction and strength of a trend. One of the most well-known trend indicators is the Moving average convergence Divergence (MACD). The MACD line is calculated by subtracting the 26-period Exponential Moving Average (EMA) from the 12-period EMA. A nine-day EMA is then applied to the MACD line, resulting in the ”signal line”. Both lines are plotted on top of each other and traders can use their interactions as buy and sell signals. A sell signal is generated when the MACD crosses above its signal line and a sell signal is generated when MACD crosses below the signal line.

2 **Momentum indicators**

By comparing the current closing price to the previous closing prices, Momentum indicators try to measure the speed at which the price is changing. One of the most popular momentum indicators is the Relative Strength Index (RSI). It measures the velocity and magnitude of recent price changes to evaluate an asset as overbought or oversold. RSI is calculated as follows:

$$RSI = 100 - \frac{100}{1 + RS} \tag{2.1}$$

$$RS = \frac{AverageGain}{AverageLoss} \tag{2.2}$$

where the Average Gain and Average Loss are commonly measured over the last 14 periods. This equation results in a value between 0 and 100. The traditional interpretation of RSI is that values above 70 indicate an overbought market, indicating that a corrective pullback in price is likely, while values under 30 imply that the market is oversold and, as such, a price increase is to be expected. Figure 2.4 is an example of how to interpret the RSI and MACD indicators.

Another well-known momentum indicator is the Stochastic Oscillator, which compares the current point of an asset's price concerning its past price range in a given time frame. The Stochastic Oscillator follows the following formula:

$$\%\,\mathrm{K} = \left(\frac{\mathrm{C} - \mathrm{L14}}{\mathrm{H14} - \mathrm{L14}}\right) \times 100 \tag{2.3}$$

where C is the current price, $L14$ is the lowest price traded in the last 14 periods and $H14$ is the highest price traded in the last 14 periods.

3 Volatility indicators

These indicators evaluate the rate at which the price is changing, regardless of direction. Introduced by J. Welles Wilder in 1978 in his book "New Concepts in Technical Trading Systems", Average True Range (ATR) is one of the indicators that fall into this category and is fairly simple to understand and compute. We start by computing the True Range, which is equal to the largest of the following three calculations:

$$CPH - CPL \tag{2.4}$$

$$CPH - PPC \tag{2.5}$$

$$PPC - CPL \tag{2.6}$$

where **CPR** = Current Period High, **CPL** = Current Period Low, and **PPC** = Previous Period Close.

The ATR is subsequently obtained by calculating an Exponential Moving Average with the True Range values. The original indicator presented by Wilder used a 14-day ATR, but this can be adjusted to the trader's preference.

4 Volume indicators

Oftentimes, an asset's price will see a rapid change that isn't supported by trading volume. Move-

ments of this type are usually followed by a price correction and, therefore, it is important that we identify them. Volume indicators measure the strength of a trend based on the volume of shares traded. A straightforward example of a volume indicator is the Volume Rate of Change (V-ROC). To calculate V-ROC, simply divide the volume change over the last n-periods (days, weeks, or months) by the volume n-periods ago. This results in a percentage that corresponds to the rate of change in volume over the last n-periods. Evidently, if the volume today is higher (lower) than the volume n-periods ago, V-ROC will be a positive (negative) number.

Figure 2.4: MACD and RSI interpretation in a BTC/USD chart

2.1.9 Blockchain

Most cryptocurrencies, including bitcoin, run on blockchain technology which was first presented to the world by Satoshi Nakamoto in 2009 [1]. At its core, blockchain is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way [19]. Many different blockchains have since emerged, offering new and different functionalities and, therefore, generating distinct information. In this work, we analyze the work done by previous authors regarding three cryptocurrencies and their corresponding blockchains:

- Bitcoin
- Ethereum
- Litecoin

.

These three are well-known cryptocurrencies whose blockchains have been studied by various authors, having achieved different levels of success. The literature around this topic was explored, but given the results observed, no blockchain-based features were tested in this work.

Lesser-known blockchains are still unexplored in the literature, however, it would be hard to design a fair trading simulation with lesser know cryptocurrencies as they often lack liquidity in a real environment, which is why they were not explored in this work.

2.2 Literature Review and State-of-the-art

Although the use of technical analysis in an effort to predict price movements has been thoroughly studied in the literature, the study of machine learning techniques applied to predicting the cryptocurrency market only started in more recent years. As such, the literature regarding some domain-specific topics such as blockchain-based features is still scarce. Nevertheless, various works made important contributions that will be reviewed more thoroughly in the following subsections. Table 2.2 provides a summary of the most relevant works mentioned in the state-of-the-art.

2.2.1 Cryptocurrency market efficiency

Fama [40] and the follow-up paper by the same author [41] introduced the Efficient Market Hypothesis (EMH), where the author states that financial markets are efficient, meaning that prices reflect all available information rendering it impossible to beat the market. This conclusion has been questioned and disputed frequently in the literature. Regarding this work's object of study, a few authors have discussed the cryptocurrency market's efficiency and its correlation to traditional asset classes such as stocks, bonds, oil, gold, and other commodities. Regarding efficiency, recent works by Andrew Urquhart [23], Bariviera et al. [24] and Vu LeTran et al. [25] share the conclusion that the cryptocurrency market is significantly inefficient, however, it has been moving towards efficiency. The correlation between the cryptocurrency market and traditional assets has been found to be weak by multiple authors over the years. Baur et al. [26], Bouri et al. [27], and Pyo et al. [28] all found weak connections between cryptocurrencies and traditional assets.

2.2.2 Data Resampling

In their work, Borges et al. [14] tested multiple machine learning models and two resampling strategies: amount resampling and percentage resampling. Various technical indicators were computed on the resampled data-sets and used as features. One of the consequences of using the resampled data was that the algorithm was significantly more active during high-volume periods, something that proved to

be more profitable. The conclusion was that regardless of the utilized learning algorithm, the outcome of utilizing resampled data consistently generated significantly higher returns than the traditionally used time-sampled data.

The resampling that obtained the best results was the percentage-based resampling. With this strategy, the authors grouped consecutive candles whose cumulative sum of absolute price variation would be equal to or greater than a specific threshold. This work will introduce a variation of this resampling technique, by taking the real price percentage variation between candles. This new technique will be further explained in Section 3.3.1.

2.2.3 Blockchain-based features

As this problem differs from traditional stock market trading, we initially intended to take advantage of domain-specific information through blockchain-based features. Although the literature on this topic is still scarce, a few authors have attempted to use blockchain-based features having achieved different levels of success.

Jaquart et al. [29] combined minutely data from four major categories - technical, asset-based, sentiment-based, and blockchain-based. The last category included the number of bitcoin transactions and the growth of the mempool (a cryptocurrency node's mechanism for storing information on unconfirmed transactions). The results indicated that the technical features were the most important for all prediction horizons (1, 5, 15, and 60 minutes). However, the relative importance of this category decreased from 80% in the 1-min horizon to less than 50% in the 60-min horizon where blockchain-based and sentiment-based features gained relevance. It is also pertinent to note that the best accuracy results were obtained for the larger prediction horizons, with a maximum accuracy of 56% for the 60-min horizon and 52% for the 1-min horizon.

Ji et al. [30] fed various deep learning models with 18 blockchain-based features and also the bitcoin price time series. The results were not promising with the models yielding a maximum accuracy of 53% and barely making a profit despite the fact that the authors did not consider trading fees. Given the high transaction fees charged in cryptocurrency markets, it is fair to question whether Ji et al.'s system would result in positive returns. Despite these results, this work brought to light an interesting finding regarding the contrast in performance between classification and regression-based algorithms, which will be later discussed.

Sebastiao et al. [34] used ensemble learning methods and achieved an annualized return of 0.5%, 9.62%, and 5.73%, when trading ethereum and litecoin respectively, during a bear market with daily mean returns lower than -0.20% and considering trading fees. The ensembles included linear models, Random Forest, Support Vector Machine, and their binary counterparts. Regarding the features used, each model was optimized in the validation set by testing different sets of features. Around 1/3 of the

models used network-based features. On the other hand, all models used the lag returns of the tested cryptocurrencies, the day-of-the-week dummies (as the name implies, this feature provides information about the day-of-the-week), and the lagged volatility proxies proving the importance of these three features.

Given the mixed results observed in the papers described above, blockchain-based features were not further explored.

2.2.4 Machine Learning Algorithms

Various machine learning approaches have been applied to financial markets forecasting, some with better results than others.

As explained previously, machine learning problems are better suited for classification tasks while others are better modeled as regression-based tasks. Therefore, it is important to start by understanding whether our problem should be approached as a classification or a forecasting task.

The previously mentioned authors, Ji et al. [30] tested the performance of multiple state-of-the-art deep learning approaches and compared the results for classification and regression-based algorithms. The outcome was strongly in favor of classification as every classification-based deep learning model beat its regression-based counterpart. The authors concluded that classification models were more effective than regression models for algorithmic trading. Therefore, in our work, we will be using classification-based models.

When it comes to the state-of-the-art in the machine learning field, deep learning takes the spotlight. But despite the great results obtained by deep learning techniques in fields of research like Natural Language Processing and Computer Vision, research regarding their application to Financial Markets has seen mixed results.

For instance, in the same study mentioned above, Ji et al. [30] concluded that, given the poor results obtained by all the tested models, it is still premature to solely use deep learning models for algorithmic Bitcoin trading. In another paper, Kwon et al. [31] used a Long Short Term Memory (LSTM) and five features: open price, close price, high price, low price, and volume at each time epoch (i.e., every 10 minutes) as input to predict future price movements. The author found that, regardless of the tested cryptocurrency, LSTM always outperformed the Gradient Boosting model used as a comparison. For reference, the LSTM's F1-score was approximately between 63 and 68% for the seven cryptocurrencies tested, while GB obtained an F1-score between 59 and 63% approximately. Alessandretti et al. [32] compared the performance of Gradient Boosting Decision Trees and an LSTM model tasked with predicting daily price variations of different cryptocurrencies. The authors found that the simpler models based on gradient boosting decision trees achieved the best results for short 5 to 10 days input windows. When dealing with larger 50 days windows, the LSTM model outperformed the simpler models. Souza et

al. [33], with a similar task and input that included open, high, low, and close prices of Bitcoin, Gold, and Silver, compared the performance of an Artificial Neural Network and a Support Vector Machine. The results showed that, when predicting bitcoin price movement, the SVM model produced similar positive results regardless of the sample period tested with a hit-rate (percentage of profitable trades) between 57.32 and 59.75% for all three time samples. On the other hand, the Artificial Neural Network (ANN) model proved to be more inconsistent having poor results for some periods but generating better returns in others. The ANN's hit-rate was between 51.38 and 61.73% for all time samples when predicting bitcoin's price movement. The authors conclude that the SVM strategy should be used by investors willing to achieve more conservative returns on a risk-adjusted basis but the ANN proved that it can generate greater profits during short bull runs. Nevasalmi et al. [36] also compared an ANN model to several other simpler models in a 3-class classification task. The clear winner was the tree-based Gradient Boosting Machine (GBM) with the ANN model coming at a distant second place and having only slightly better results than the other tested models.

On the other hand, ensemble models seem to come out on top consistently in works that provide a comparative analysis between models. Borges et al. [14] tested the performance of two linear methods - Logistic Regression and Support Vector Machine, two non-linear methods - Random Forest and Decision Tree Gradient Boosting, and an ensemble of these 4 algorithms. The results showed that the ensemble outperformed every other algorithm in both return on investment and accuracy. Sebastião et al. [34] tested Linear models, Random Forest, Support Vector Machine, and their binary versions, in a total of 6 individual models. Three ensembles were then built, based on three different voting thresholds (each model required at least 4, 5, or 6 votes to enter a long position). The conclusion was that the ensemble beat the individual models and also the Buy and Hold strategy that was used as a benchmark. Mallqui et al. [35] also found that model assembling yielded positive outcomes when predicting price movements in the cryptocurrency market.

2.2.5 Multinomial Classification

In a recent paper, Nevasalmi [36] proposed a new non-binary classification approach to forecasting stock returns. Two thresholds c1 and c2, defined as the upper and lower quartiles of the return series, separate three possible classes. The idea is to isolate the noisy fluctuation around zero (between c1 and c2) and focus on the larger changes in value. Therefore, price movements between c1 and c2 are considered noisy fluctuation whereas price movements outside these bounds are considered significant. Furthermore, this approach also allows for a more elaborate trading strategy as the author states.

In this work, this approach was not tested as it was not compatible with our strategy. However, a slight modification is proposed for future work - instead of the upper and lower quartile, the data sets tertiles (two points that divide an ordered distribution into three parts, each containing a third of the population)

should be used. This would change the class distribution from (1/4, 1/2, 1/4) to (1/3, 1/3, 1/3) making it a balanced data set. This also could also result in a more active trading strategy, as smaller price movements will be considered significant.

Ref.	Year	Financial Market	Used methodologies	Performance	Main contribution
[14]	2020	Cryptocurrency	LR, RF, GTB, SVM, EV	≈55% acc., ≈1000% ROI,	Volume and percentage resampling
[29]	2021	Cryptocurrency	NN, FFNM LSTM and GRU, RF, GTB, Ensemble	≈56% acc.	Blockchain-based data had more influence on large prediction horizons
[30]	2019	Cryptocurrency	DNN, RNN, LSTM, CNN, DRN, Ensemble	≈53% acc.	Classification beats regression for algorithmic trading
[31]	2019	Cryptocurrency	LSTM, GB	≈68% F1-score	LSTM outperformed GB
[32]	2018	Cryptocurrency	LSTM and GTB	Signifcant profit even with transaction fees up to 0.2%	GTB outperformed LSTM for shorter 5-10 days windows while LSTM performed better for 50-days windows
[33]	2019	Cryptocurrency	SVM, ANN	≈62% hit-rate	SVM had good results regardless of the chosen time-frame, while the ANN had poor results for some periods but abnormal returns during bull-runs.
[34]	2021	Cryptocurrency	SVM, RF, Linear models	Annualized returns or 9.62% with 0.5% trading fees	Ensemble outperformed its individual parts
[35]	2019	Cryptocurrency	ANN, SVM, Ensemble	62.9% acc.	Model assembling yielded positive outcomes
[36]	2020	Stocks	KNN, GBM, RF, ANN, SVM	≈56% acc.	Multinomial approach

Table 2.2: Summary of the most relevant works mentioned in the state-of-the-art.

3

System Architecture

Contents

In this chapter, we explain in detail the proposed solution to our bitcoin trading system. First, a brief overview of the system's main architecture is given. Then, the following subsections detail each module more thoroughly, in the same order as data flows through our system - from data extraction to the trading simulation.

3.1 System Overview

Our system's architecture consists of four layers, composed of five modules in total as listed below.

1. Data extraction (Layer 1)
2. Feature engineering and resampling (Layer 2)
3. Hyper-parameters tuning (Layer 2)
4. Prediction (Layer 3)
5. Trading (Layer 4)

Figure 3.1 is a representation of the flow of information in our proposed solution. As illustrated, each layer's output will be the next layer's input.

In the data extraction module, we simply collect the 1-minute candles from 2018-01-01 to 2022-01-27 (49 months), directly from Binance's API. This data includes, for each candle:

1. Opening time
2. Opening price
3. Closing price
4. Highest price
5. Lowest price
6. Volume
7. Number of trades done

In the next module (Feature engineering and resampling), we start by calculating some technical indicators, like the RSI and MACD, on the 1-minute candles. The new table is then resampled by percentage-change - the candles are grouped into larger candles that represent absolute price movements of around 4%. During this aggregation, metrics like mean, maximum, minimum, and standard deviation of each variable are computed for each resampled candle. Afterward, the final technical and time-based features are computed from transformations on the base features, listed in Table 3.1, like exponentially

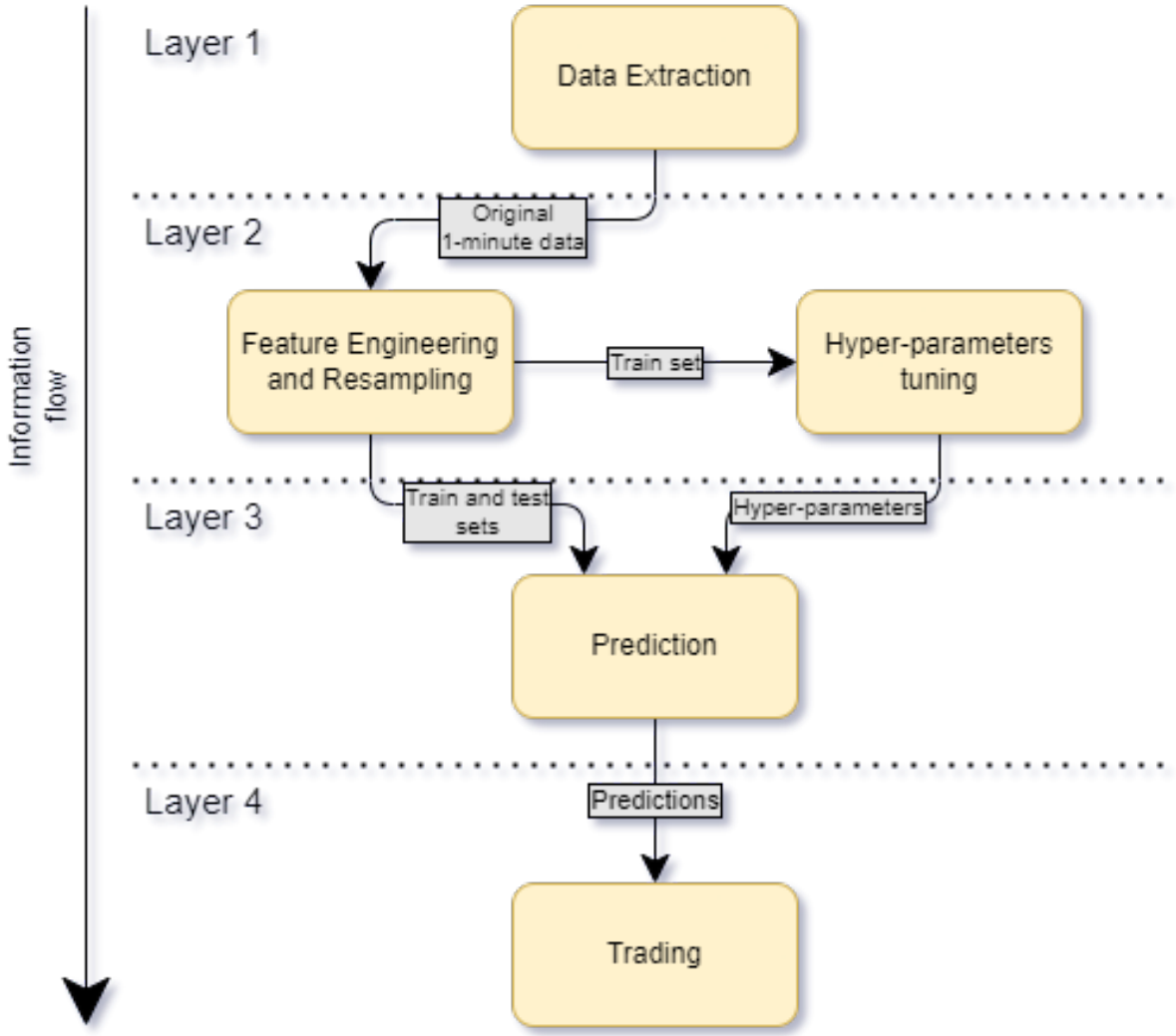

Figure 3.1: System architecture overview

weighted moving averages, percentage variations from the previous candle to the current one, or divisions by ”Delta” (elapsed time) to understand the rate of change. As this is a time-series problem, lagged features are also computed. The full list of operations performed on the base features follows:

- Column shifts to create lagged variables;
- Difference (from the last candle to the current one);
- Percent variation (from the last candle to the current one);
- Rolling average;
- Exponentially weighted average;

- Division by Delta;
- Division by Close price;
- Negative and positive candle amplitude.

The features that result from these operations are identified by a suffix, that corresponds to the performed operation, preceded by the name of the affected base feature. Figure 4.2 shows the most important features, ranked by the number of splits where each feature was present in the XGBoost model.

Because we are testing distance-based algorithms like Support Vector Machine and Logistic Regression, the computed features are then normalized to the [0,1] range, using a Min-Max Scaler.

Next, the training and test sets are established following a holdout set strategy, where the first 80% of data points were assigned to the train-set, and the final 20% belonged to the test-set. In more practical terms, this means our test-set was a period of 284 days, between 2021-04-18 and 2022-01-27.

Finally, the training set is re-balanced so we have the same amount of records in the positive and negative classes. As we had enough training data on the minority class, we decided to under-sample the majority class using a random under-sampling technique.

The Hyper-parameters tuning module receives the training set from the previous module, and, for each algorithm, it determines the best hyper-parameters for the type and amount of data we have. This is done through Bayesian optimization, using stratified K-Fold cross-validation.

The recently computed features and hyper-parameters are then passed on to the prediction layer. Here, various machine learning algorithms learn by looking at the data in the training set and trying to predict the label for all data points in the test set. The machine learning algorithms present in this layer are listed below:

- Logistic Regression
- Support Vector Machine
- Random Forest
- Gradient Tree Boosting
- Ensemble - composed of three of the previous algorithms

A more detailed description of each of these algorithms is present in Section 3.5.

The predictions are then passed through to the trading layer where a trading simulation is performed. We use the binary predictions to decide when to enter and leave the market - if the next candle is predicted to be positive, we either enter or stay in the market; otherwise, we either sell or stay out

depending on our current position. With every transaction, the whole value of our portfolio is used. This means, at any given time, our portfolio is composed of either 100% bitcoin or 100% money.

A more detailed description of each of these modules follows.

3.2 Data Extraction

In this module, the data is extracted from a publicly available API from Binance. As mentioned in the system architecture overview, the raw data obtained from Binance will contain only the following metrics for each candle:

1. Opening time;
2. Opening price;
3. Closing price;
4. Highest price;
5. Lowest price;
6. Volume;
7. Number of trades completed.

As the data will later be resampled, there is a need to obtain it with the finest granularity possible. The finest granularity offered by Binance is one sample per minute. For this reason, the original time-sampled data will have one data point per minute.

The data collected pertains to the period between 2018-01-01 and 2022-01-27 (49 months). Despite bitcoin being introduced much earlier, Binance was only founded in the second half of 2017, thus, earlier data was not available.

3.3 Feature engineering and resampling

In this step, the original data is resampled following a variation from the original approach from Borges et al [14], as explained in detail in the next Section.

Time-based and technical features are also computed in this step. The RSI and MACD features are computed before the resampling, while the others are computed after the resampling.

The target variable is also computed: each new data point gets labeled according to the price movement observed in the next candle. If the price movement, in percentage, is negative, then it will belong to class 0. Otherwise, if it is positive, it will belong to class 1.

3.3.1 Resampling

The 1-minute time-sampled raw data is passed on to this module. Here, groups of consecutive candles that represent either positive or negative price movements above a certain threshold are aggregated to form single candles.

As previously mentioned, this step is loosely based on Borges et al.'s percentage-based technique. There are two main differences:

1. The price variation is calculated as the price difference (in percentage) between the first and last candle pertaining to the same resampled candle (as opposed to the cumulative sum of every candle's absolute volatility);

2. The technical features are computed on the original time-sampled data and then aggregated through averages (as opposed to being computed on the resampled data).

So while the previous technique's resampled candles contained similar absolute cumulative volatility, they could represent a wide range of price variations. On the other hand, the resampling technique used in this work, provides resampled candles that can represent different levels of absolute cumulative volatility but they always represent the same absolute price variation.

Initially, the price variation threshold (resampled candle size) was set at 4% for both positive and negative movements. But this proved not to be the optimal solution, as percentage variations are not symmetrical - e.g. a 10% price variation, followed by a -10% drop will leave us at a loss - which means a system at around 50% prediction accuracy will most likely not be profitable, even if we disregard the transaction fees. This led us to find the formula that establishes the relation between positive and negative price changes. In other words, given a positive price change, we can compute the corresponding negative price change that will lead to a break-even position, when disregarding transaction fees. The formula follows:

$$negVariation = 1 - \frac{1}{1 + posVariation} \tag{3.1}$$

Substituting *posVariation* by 0.04 (4%) in the above formula leads to y $\approx$ 0.0384 (3.84%), which was the threshold used for the negative price variations. This small change consistently improved our results regardless of the positive variation chosen. Figure 3.2 shows this asymmetry for a few values. Both the utilization of asymmetric candles and the optimal resampling threshold size were thoroughly analyzed and are presented as the second and third case studies in Section 4.3.

When aggregating the previous 1-minute time-sampled candles, it is important to minimize information loss. Each new candle will be described by:

- Opening time - The first candle's Opening time;
- Open - The first candle's Opening price;

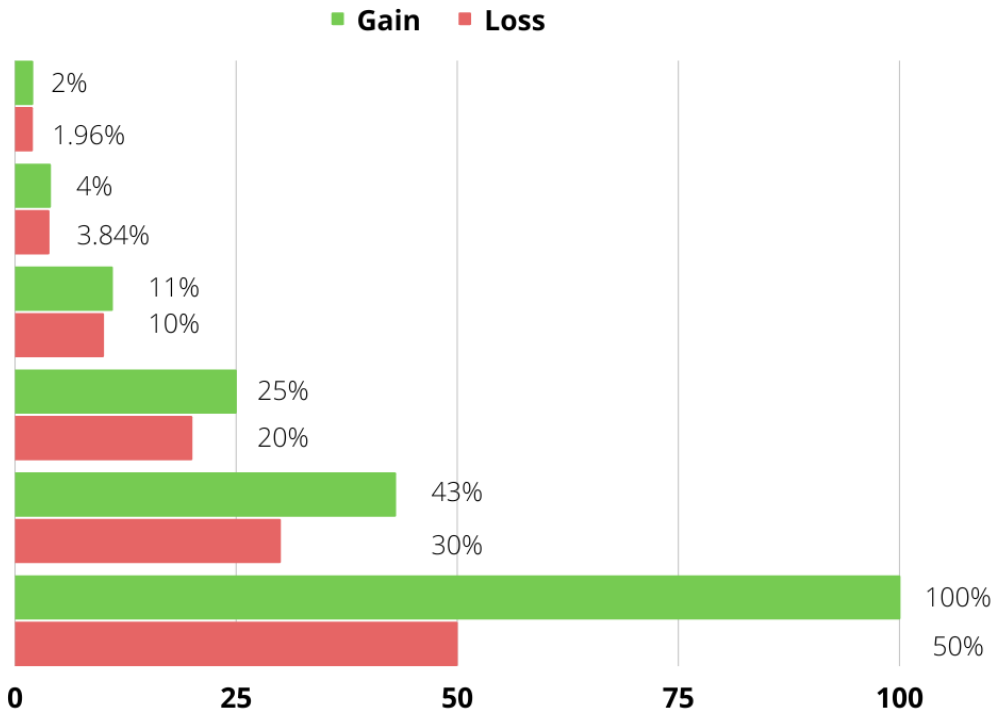

Figure 3.2: Break-even gains and losses

- Close - The last candle's Closing price;
- High - The highest price registered in all candles;
- Low - The lowest price registered in all candles;
- Volume - The sum of the volume of all candles;
- Number of trades - The sum of the number of trades of all candles;
- RSI - Mean RSI values;
- MACD - Mean MACD values;
- Delta - The time elapsed between first candle's Opening time and the last candle's closing time (opening time + 1 minute).

One last thing to note about this resampling is that, because the threshold is computed on the 1-minute time-sampled candles closing value, this means the price variation of the resampled candle will not always be exactly 4% or -3.84%, most times it will have an absolute value slightly above these numbers, but never below them.

Figure 3.3 shows an example, for illustrative purposes, of a resampling procedure, both using the approach from Borges et al. [14] and also this work's approach. In this table, each separate coloured group will form a new candle, corresponding to its group number.

Figures 3.4, 3.5 and 3.6 show the original time-sampled candles, the Borges et al. [14] absolute percentage-based candles and this work's percentage-based candles respectively, based on the example given in Figure 3.3. One thing to note is that the time-sampled candles represented on this example intentionally contain very large price movements, so that the resampling would be easier to observe. Additionally, the considered thresholds were 4% for Borges et al.'s [14] resampling and 3% for this work's resample, which do not represent the real thresholds used in either work.

As explained earlier, it is easily observed in Figure 3.5 that the candles provided by Borges et al.'s [14] resampling may lead to candles with no significant price change (candle 4), but always with similar total absolute cumulative volatility. While in Figure 3.6, we can see that the candles provided in this work always contain a significant price movement and, when applied to real 1-minute candles that have lower volatility, most candles will represent a similar absolute price change.

We believe our approach is more beneficial as it aligns the metrics that the models are optimized for (accuracy) with the system's objective (being profitable): with Borges et al's resampling, when training a model, an incorrect prediction on a 0.1% price variation candle will have the same weight as an incorrect prediction on a 5% price variation candle, although the latter is clearly more important than the former, when trying to create a profitable trading strategy. With the resampling proposed in this work, each data point will hold similar absolute price variations, thus, optimizing accuracy will presumably optimize returns.

As a final note, we believe the resampling introduced in this work also lines up better with the message that the technical indicators convey: an RSI value above 70 does not indicate that the price will drop in the next time-period (as implied when using traditional time-sampled data and predicting the next candle), but a high RSI over a few candles does suggest that the next significant price movement should be a negative one.

3.3.2 Feature engineering

The final base features to be considered, described in Table 3.1, are then computed on the resampled data. These features first went through a correlation test to understand if they might have predictive power over bitcoin's price movement. Correlation between features was also computed, as features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable (the cryptocurrency's price movement). The presence of two or more highly correlated variables in the training set might lead our models to overfit the specific characteristic that those variables describe, which should be avoided.

Index	Open	High	Low	Close	Absolute variation	Cumulative sum *	Assigned group (Borges et al.)	Price variation *2	Assigned group (this work)
1	23 202	23 202	20 808	20 831	10,22%	**10,22%**	1	**10,22%**	1
2	20 831	21 357	20 785	21 139	1,48%	1,48%	2	1,48%	2
3	21 139	21 692	21 077	21 517	1,79%	3,27%	2	**3,29%**	2
4	21 517	21 517	20 912	21 416	0,47%	3,74%	2	-0,47%	3
5	21 417	21 661	20 920	21 517	0,47%	**4,21%**	2	0,00%	3
6	21 516	21 832	21 171	21 365	0,70%	0,70%	3	-0,71%	3
7	21 365	21 801	21 325	21 565	0,94%	1,64%	3	0,22%	3
8	21 564	21 815	20 122	20 250	6,09%	**7,73%**	3	**-5,89%**	3
9	20 250	20 344	19 850	20 034	1,07%	1,07%	4	-1,07%	4
10	20 034	20 151	19 543	19 550	2,42%	3,48%	4	**-3,46%**	4
11	19 551	20 395	19 551	20 296	3,81%	**7,29%**	4	**3,81%**	5
12	20 296	20 558	19 560	19 793	2,48%	2,48%	5	-2,48%	6
13	19 793	20 469	19 793	20 044	1,27%	3,74%	5	-1,24%	6
14	20 050	20 203	19 584	20 126	0,38%	**4,12%**	5	-0,84%	6
15	20 132	20 428	19 765	19 953	0,89%	0,89%	6	-1,69%	6
16	19 953	20 044	19 661	19 831	0,61%	1,50%	6	-2,29%	6
17	19 832	20 018	19 595	20 000	0,85%	2,34%	6	-1,46%	6
18	20 000	20 043	19 651	19 793	1,03%	3,38%	6	-2,48%	6
19	19 793	20 169	19 687	19 894	0,51%	**3,89%**	6	-1,98%	6

Figure 3.3: Resampling example.
* Group closes when 5% threshold is exceeded.
*2 From current candle to last candle in previous group, group closes when 3% threshold is exceeded.

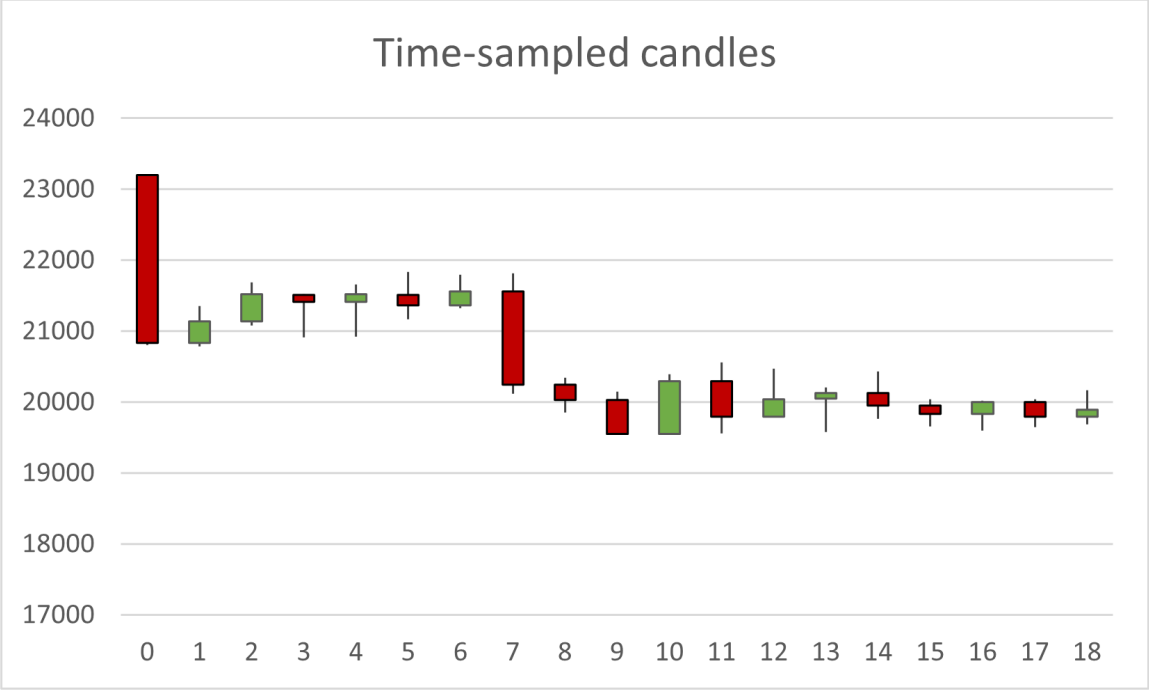

Figure 3.4: Original time-sampled candles

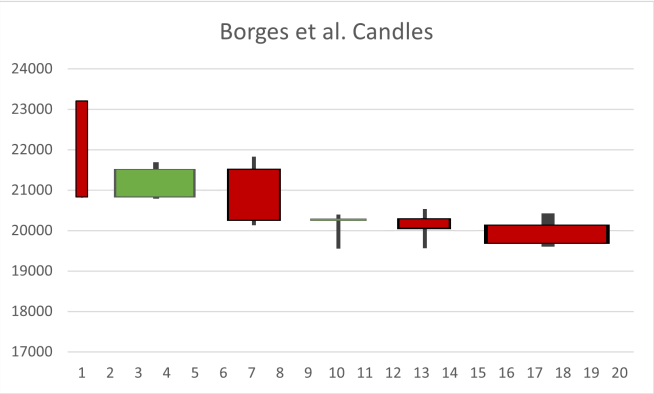

Figure 3.5: Borges et al. [14] candles

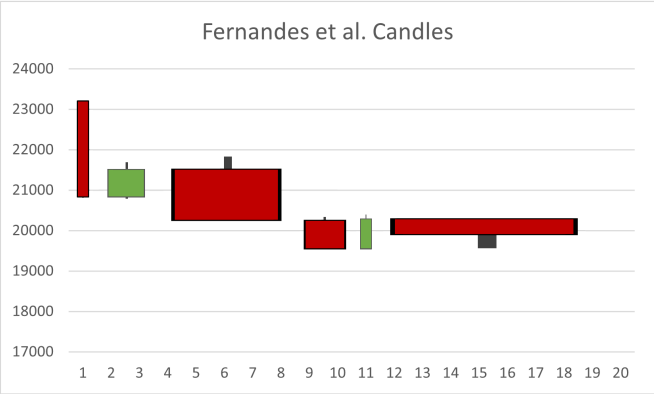

Figure 3.6: Fernades et al. candles

As mentioned before, two types of features were considered - technical and time-based. The former were computed before the resampling, and for each new candle, the mean values of the original candles were computed. On the other hand, the time-based features were computed after the resampling, with the intent of maintaining the notion of time within our data-set.

From the base features computed in the last step, which are listed in Table 3.1, multiple other features were computed through data manipulation or arithmetic operations.

First, lagged features - features that refer to previous candles - were computed by simply shifting our dataframe's columns by the desired number of time-steps.

Secondly, features based on the difference and percentage of change between the lagged features and the current ones were computed. For each candle, the negative and positive amplitudes were also computed.

Then, a few different moving averages and exponential rolling averages were computed, by varying the window size and the decay in terms of center of mass respectively.

Lastly, some features were divided by the Closing price and others by the candle's Delta (elapsed time), in order to obtain the rate of change of said feature. This last operation was performed on the features that pertained to Volume, Price change, or Number of trades. The full list of operations performed on the base features follows.

- Column shifts to create lagged variables;
- Subtraction (from the last candle to the current one);
- Percent variation (from the last candle to the current one);
- Rolling average;
- Exponentially weighted average;

- Division by Delta;
- Division by Close price;
- Negative and positive candle amplitude.

These operations were also performed sequentially in specific cases (e.g. the percent variation between the Closing price rolling average from the last candle to the current one).

Too many features were computed to analyse each of them individually, but the 15 most important ones listed in Figure 4.2 (as computed by one of the tree-based models, based on the number of splits each feature was utilized on) will be analysed in Section 4.2.

Feature	Description
Close	Closing price.
Open	Opening price.
High	Highest price.
Low	Lowest price.
Num_Trades	Total number of trades.
RSI_14	Average RSI, with each base indicator calculated over fourteen 1-minute candles.
RSI_60	Average RSI, with each base indicator calculated over sixty 1-minute candles.
MACD	Average MACD, calculated on 1-minute candles.
MACD_Signal	Average MACD signal, calculated on 1-minute candles.
MACD_Sub	Average difference between MACD line and MACD signal, calculated on 1-minute candles.
Var_Percent	Percentage variation between the previous Closing price and the current one.
Max_price	Maximum price achieved by bitcoin up to the given point.
Max_volume	Maximum volume achieved by bitcoin up to the given point.
Delta	Time elapsed during the current candle.

Table 3.1: Base features.

3.3.3 Data preprocessing

The final step in this layer is to transform the recently computed variables into usable features. Data scaling and variable encoding were performed in this step.

As explained previously, because all these features fluctuate between a different range of values, and because distance- and Gradient Descent-based algorithms were used, it was necessary to scale

our variables. To achieve this, the Min-Max scaler from the sklearn package was used. Min-Max uses the following formula to scale the features:

$$x_{new} = (x - x_{min})/(x_{max} - x_{min}) \tag{3.2}$$

The new features will range from 0 to 1, thus eliminating geometry bias that would affect the SVM and Logistic Regression algorithms.

The final preprocessing step is variable encoding. As explained in the Background Section, not all algorithms are ready to deal with categorical variables. Therefore, these variables need to be translated into dummy numeric variables. The only categorical variable tested in this work was the day of the week, which was encoded using the Sine and Cosine functions as detailed in the Data Preparation subsection. However, this variable did not prove to have any correlation with bitcoin's price movement and, therefore, it was not further analyzed.

3.3.4 Training and Test sets

As previously mentioned, a holdout strategy was followed to evaluate our system. Therefore, the pre-processed data-set was split into two data-sets: one for training the model - the training set, and another one for testing its predictions on unseen data - the test set. The training set contained the first 80% of data points, while the test set was composed of the last 20% of data points.

Given the nature of this problem, the training-set points must be taken sequentially from the beginning of the data-set, so as not to give our models future information about Bitcoin's price movements. The considered train-set contained data from 2018-01-01 to 2021-04-18 (39.5 months) and the test-set data ranged from 2021-04-18 to 2022-01-27 (9 months).

The final step in this layer is balancing the training-set classes. Having unbalanced classes (a class with more records than the other) might lead to a biased model, especially in a problem where the model's accuracy is expected to be close to 50%. To avoid this, we resorted to undersampling with the RandomUnderSampler function from the package imblearn. As the name suggests, this function selects a random sample from the majority class with the same number of records as the one in the minority class, thus balancing both classes.

Now that the features are ready and the training and test sets were separated, we can pass the training set on to our next layer where the hyper-parameters of the various machine learning algorithms will be optimized.

3.4 Hyper-parameter tuning

This module only receives the training set, so as to avoid data leakage.

The optimization of hyper-parameters plays a vital role in maximizing the performance of a machine learning model, especially when looking at algorithms that have a significant number of hyper-parameters that need tuning like XGBoost and Random Forests.

The most common strategies for this task are Grid Search and Random search. Both of these strategies and also Bayesian optimization through Hyperopt, try to find the best values in a given search space. A search space consists of the variables to be optimized and the possible values these variables can assume. As the names imply, Grid Search works by trying every possible combination of hyper-parameters given in the search space, while Random search tries random combinations of values. Both of these have very clear drawbacks: Grid Search is very computationally expensive, while Random search can miss important combinations of values.

For the reasons above, Bayesian optimization through Hyperopt was used in this work. Bayesian optimization is a probabilistic model-based technique used to find the minimum of any function. It takes into account past evaluations when choosing the next combination of hyper-parameters to try. In other words, it focuses on value combinations that yield the best possible scores in past iterations. Therefore, it chooses its hyper-parameter combinations in an informed way and can rapidly learn to focus on a smaller section of the search space, where the results are best. This technique requires a smaller number of iterations to find the optimal set of parameter values as it ignores areas of the parameter space that are useless.

To find the optimal parameters, a few rounds of Bayesian optimization were performed, with the total number depending on the Machine Learning (ML) algorithm being optimized and its complexity. First, a more general tuning round was done with some variables containing values in different orders of magnitude. For example, the search space for the variable *reg_alpha* for XGBoost started with the following range of values: [0.001, 0.01, 0.1, 1, 10, 50, 100]. The optimal value here was 10, so a new round of Bayesian optimization was done around this value until we reached the final value of 9.25. For simplicity, only one variable was mentioned in the previous example, but in reality, all the hyper-parameters for a given algorithm were being optimized at the same time, to ensure we have the optimal combination of hyper-parameter values, rather than a combination of optimal individual values.

As mentioned before, Bayesian optimization minimizes a function. The minimized function was the stratified K-fold validation accuracy score (with k=10), multiplied by negative one (so that minimizing this value implies better accuracy).

Table 3.2 shows the tuned hyper-parameters, their description, and the value used to train the final models. We now have everything ready for our models to start learning from the historical preprocessed data (training set) to make predictions about future price movements (test set).

Algorithm	Hyper-parameters	Description	Final value
Logistic Regression [docs]	C	Inverse of regularization strength.	0.152
	fit_intercept	Specifies if a constant should be added to the decision function.	False
	penalty	Specify the norm of the penalty.	'l2'
	solver	Algorithm to use in the optimization problem.	'lbfgs'
Support Vector Machine [svm docs]	C	Inverse of regularization strength.	2.17
	gamma	Kernel coefficient.	0.22
	kernel	Specifies the kernel type to be used in the algorithm.	'rbf'
Random Forests random forest docs	bootstrap	Whether bootstrap samples are used when building trees.	True
	criterion	The function to measure the quality of a split.	'entropy'
	max_depth	The maximum depth of the tree.	7
	max_features	The number of features to consider when looking for the best split.	'sqrt'
	max_samples	The number of samples to draw from X to train each base estimator.	0.88
	min_samples_leaf	The minimum number of samples required to be at a leaf node.	7
	n_estimators	The number of trees in the forest.	40
XGBoost [xgb docs]	colsample_bytree	The subsample ratio of columns when constructing each tree.	0.6
	subsample	Subsample ratio of the training instances.	0.83
	learning_rate	Step size shrinkage used in update to prevent overfitting.	0.18
	max_depth	Maximum depth of a tree.	13
	min_child_weight	Minimum sum of instance weight (hessian) needed in a child.	10
	reg_alpha	L1 regularization term on weights.	9.25
	reg_lambda	L2 regularization term on weights.	12.25
	n_estimators	Number of gradient boosted trees.	90

Table 3.2: Final hyper-parameters after multiple rounds of tuning.

3.5 Prediction module

The prediction layer receives the optimized hyper-parameters, the training-set, and the test-set. The first two are utilized to train each model and, after the training phase is complete, the test set is received by the trained model which outputs a price movement prediction between two classes.

As mentioned earlier, a comparative analysis between multiple machine learning models will be provided. Given the results observed in the literature and their reduced required computing power, the focus of this work was on traditional machine learning models and ensembles. Following, a description of the algorithms tested in this work is given.

3.5.1 Logistic Regression

Logistic regression is very similar to linear regression, with the exception that it predicts discrete values, instead of real ones. It is commonly used in binary problems, such as this one. It uses a logistic function defined below to map input parameters to class probabilities.

$$f(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} \tag{3.3}$$

where X is the input, β_0 is the bias or intercept term and β_1 is the coefficient for the given input X. The output of this function will always be between 0 and 1 and corresponds to the probability of the record's true class being the positive one. In this work, the threshold considered to predict a class as positive will be 0.5, not only for this algorithm but also for all others that output a probability.

Logistic regression generates a linear decision boundary, which might prove too basic for the problem we are trying to solve. Regardless, it is a good baseline for most machine learning problems and will serve as a comparison for the more complex algorithms. In this work, python's scikit-learn package will be used to implement this algorithm.

3.5.2 Support Vector Machine

A Support Vector Machine is a binary classification algorithm that works by finding the hyperplane in an N-dimensional plane that best separates the data points, where N is the number of features.

To find the ideal hyperplane, the algorithm maximizes the margin between the hyperplane itself and the support vectors (the closest points of each class).

Kernel operations determine how similar two points are, after a given linear or non-linear transformation. The original SVM algorithm is a linear classifier, but by transforming the feature space through a non-linear kernel operation, a non-linear hyperplane can be found, thus improving results in non-linear problems.

In this work, linear, polynomial, and Radial Basis Function (RBF) kernels were tested in the hyper-tuning module. The RBF kernel obtained the best results and, therefore, it was the selected kernel for the final architecture. The formula for the RBF kernel function, between two points X1 and X2, follows:

$$K\left(X_1, X_2\right) = \exp\left(-\gamma \left\|X_1 - X_2\right\|^2\right) \tag{3.4}$$

where γ is a hyper-parameter that defines how far the influence of a single training example reaches. In other words, smaller values of γ lead to a straighter line, closer to a linear model which may lead to under-fitting, while very large values may lead to a hyperplane that is influenced too far by a single training example, thus, leading to over-fitting.

In this work, python's scikit-learn package was used to implement this algorithm.

3.5.3 Decision Tree

While a Decision Tree was not directly used in this work, it is the building block of the next two algorithms (Random Forest and Gradient Boosted Trees).

As the name implies, a Decision Tree model splits the data according to a sequence of decisions. Each decision corresponds to a simple single variable condition, either in the form of a threshold for continuous variables or equality in the case of categorical variables. These decisions are made at decision nodes. Simply put, a decision tree is composed of two types of nodes - decision and terminal (or leaf) nodes. It is also usual to consider the root node, which is simply the first decision node.

When training a decision tree model, at each new node starting from the root node, the best available split is calculated by computing the Information Gain (or a similar metric) using each available variable. In other words, the algorithm tries to find the split (variable and value) that produces the most homogeneous sets.

If the information gain obtained by the optimal split is superior to a certain threshold, then the node becomes a decision node and the data is split accordingly. On the other hand, if the information gain does not surpass this value, or if the data at this node is already homogeneous, then the node becomes a leaf node.

As mentioned, decision trees are the basic building block of the Random Forest and Gradient Tree Boosting algorithms. There are, however, a few differences in the training process that will be further explained below.

Figure 3.7 shows an example of a binary classification decision tree, based on the task of predicting if the market is going up or down.

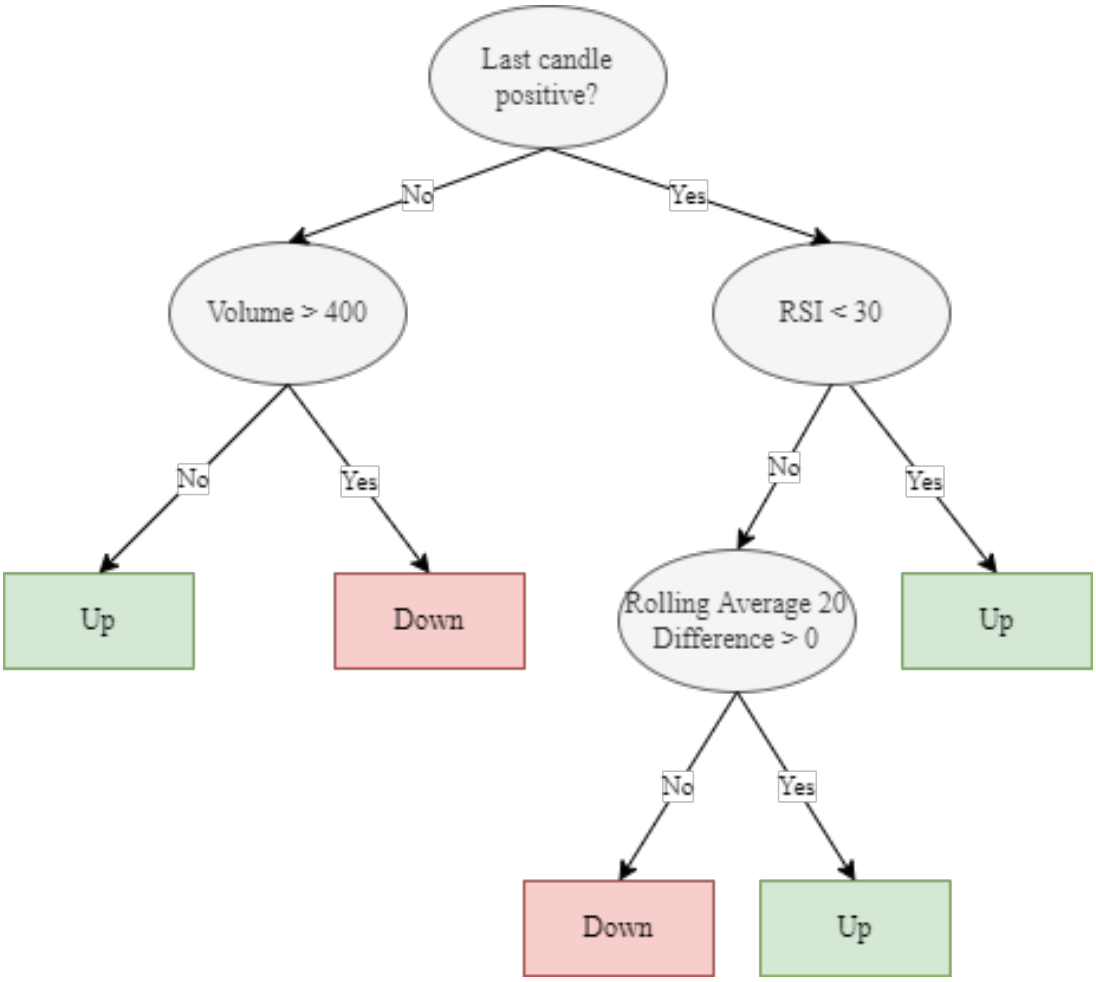

Figure 3.7: Basic decision tree

3.5.4 Random Forest

A Random Forest is an ensemble of decision tree models. This kind of approach leverages the fact that a combination of relatively uncorrelated models will outperform its constituents [39].

To optimize an ensemble's performance, the constituents must be relatively uncorrelated so that there is a higher probability that wrong individual predictions are corrected by the majority vote. To ensure that the decision trees are uncorrelated, bagging (also called bootstrap aggregating) and feature randomness were used. Both these methods work by extracting a random sample of the available data. A brief description of each of these techniques follows:

- Bagging - A random sample of data points is selected to train each tree;
- Feature randomness - A random subset of features is considered when splitting nodes. This contrasts with the basic decision tree algorithm that looks at every available feature at every split.

The percentage of data points for each tree and the ratio of features considered for each split are con-

trolled by hyper-parameters, and the optimal values for each of them were computed through Bayesian optimization, as mentioned earlier.

In this work, python's scikit-learn package was used to implement the Random Forest algorithm.

3.5.5 Gradient Tree Boosting

Gradient Tree Boosting (GTB) is similar to RF, in the sense that it combines the efforts of multiple weak learners (decision tree algorithms) in order to improve the accuracy of a strong learner which is the final model. Also, as with the Random Forest algorithm, bagging and a variation of feature randomness (here, a subset of features is considered for each tree, as opposed to each split) were also applied here to keep the tree's correlation as low as possible.

However, the main difference lies in how the trees are combined. In a random forest, the trees are combined after being created, while in GTB the trees are created and added to the model sequentially, with the end goal of optimizing an objective function. More specifically, in GTB each new weak learner is created in a way that minimizes the model's current errors estimated by the following objective function: [42]:

$$Obj(\theta) = L(\theta) + \Omega(\theta) \tag{3.5}$$

where Ł is the training loss function and Ω is the regularization term. In this work, as our task is binary classification, the loss function used was the negative log-likelihood loss function:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{3.6}$$

The XGBoost framework was used to implement this algorithm, therefore, the regularization term is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{3.7}$$

where T is the number of leaves and w is the vector of scores on leaves.

As the XGBoost package was used to implement this algorithm, the terms XGBoost and Gradient Tree Boosting were used interchangeably throughout this work.

3.5.6 Ensemble

The basic majority-vote ensemble included three of the previous algorithms - SVM, RF, and XGBoost as these had the best individual results. Each of them will predict the next price movement, which will count as a vote. A simple majority vote decides the ensemble's prediction for the next candle.

Different ensemble trading strategies were also tested by defining two different vote thresholds: one for entering the market and one for exiting the market. Lower thresholds will account for a more active trading strategy, while higher thresholds will lead to a more passive strategy with longer periods between trades.

These strategies were only evaluated by their financial performance, as having different thresholds depending on the situation (in or out of the market) does not allow for a correct evaluation with the usual machine learning metrics. Therefore, this study was only performed in Section 4.3.

3.6 Trading module

The last module receives the predictions and simulates trades accordingly, with the intention of understanding what the models' performance would be in a real market environment.

This module receives a total of 5 time-series, containing the predictions of each individual model and the basic majority-vote three-model ensemble, and creates the corresponding 5 trading strategies.

For each data-point, a prediction that the next candle will be positive is represented by a label "1", while the opposite is represented by "0". These labels are converted to trading signals by the control function *f*, defined as follows:

$$f = \begin{cases} "BUY" & \text{if not } currently\_in() \text{ and } label = 1 \\ "SELL" & \text{if } currently\_in() \text{ and } label = 0 \\ \text{"Do nothing"} & else \end{cases}$$

where $currently\_in()$ is a function that returns True if the system is currently holding bitcoin, and False otherwise. Thus, if the next candle is predicted to be positive, we either enter or stay in the market; otherwise, we either sell our position or stay out, depending on our current position.

With each transaction (buying or selling bitcoin), a 0.1% fee was considered, as this is Binance's default fee for each transaction.

As we were dealing with historical data, this simulation was done through backtest trading, which is simply the process of employing a trading strategy on past data. The implied logic here is that, if the trading strategy performs well on past data, it will likely perform well on future data too. Because our trades are only simulated, two assumptions are made in this module, the same ones mentioned by Borges et al [14] in their work:

1. Market liquidity: The markets have sufficient liquidity to complete any trade placed by the system immediately and at the current price of its placement.

2. Capital impact: The capital invested by the algorithm has no influence on the market as it is relatively insignificant.

The trading system starts every experiment with an initial sum of 1000$. First, each model was tested individually, and, afterward, a few strategies to combine the predictions of the individual models were tested.

The Buy and Hold strategy was also tested in this module, by simply making a transaction at the starting point of the simulation period, and selling at the last possible moment. This strategy will be used as a financial performance baseline in the following Section.

4

Results and Case Studies

Contents

In this Section, a thorough explanation of the evaluation methods is given, followed by the obtained results for the base models and resulting trading strategies as well as their interpretation. Lastly, three case studies are presented where a specific parameter or approach is more thoroughly analyzed, by experimenting with different values or strategies and comparing the final results.

4.1 Work evaluation methodology

The first thing to note is that, as mentioned before, the data-set was split between a train-set and a test-set. The train-set contains the data pertaining to the period between January 1st, 2018 and April 18th, 2021 (39,5 months), while the test-set contains data pertaining to the period between April 18th, 2021 and January 27th, 2022 which amounts to 284 days (9 months), which also corresponds to the trading simulation period.

Dataset	Start	End	Total Time	# Candles	Pos. candles	Neg. Candles
Train-set	01/01/2018	18/04/2021	$\sim$39,5 months	1255	649	606
Test-set	18/04/2021	27/01/2022	$\sim$9 months	314	155	159

Table 4.1: Data-set characteristics.

In terms of data points, this resampling with a 4% threshold generates a total of 1255 data-points for the train-set, and 314 for the test-set. Some data-set characteristics are summarized in Table 4.1.

The results that are presented below were obtained in a trading simulation performed on the aforementioned test-set period. Given the nature of this work, the evaluation of our model will be split into two parts:

- Model evaluation - how well the models are predicting bitcoin's price movements, as measured by traditional machine learning metrics;
- Financial evaluation - how profitable our model is, according to well-known investment metrics.

.

4.1.1 Model evaluation

Various metrics have been developed over the years to evaluate machine learning models. Understanding these metrics is key to assessing a model's performance. A confusion matrix summarizes the predictions made by a classification model. Considering one class as the "positive class" and all others as "negative classes", we can observe four key measures from the confusion matrix:

- True Positive (TP) - A correctly classified positive class instance;
- True Negative (TN) - A correctly classified negative class instance;
- False Positive (FP) - A negative class instance wrongly classified as positive;
- False Negative (FN) - A positive class instance wrongly classified as negative.

And from these concepts we can compute performance metrics that will be used to evaluate our model, described in 4.2:

Metric	Formula	Intuition
Accuracy	$(TP + TN)/(TP + TN + FP + FN)$	The percentage of correct predictions.
Precision	$TP/(TP + FP)$	The quality of a positive prediction.
Recall	$TP/(TP + FN)$	The percentage of positive records found.
Specificity	$TN/(TN + FP)$	The quality of a negative prediction.
F1-Score	$2 * Precision * Recall/(Precision + Recall)$	How correct and balanced the model is.

Table 4.2: Evaluation metrics formula and description.

In this work, we considered the positive class to be the one that represents positive price movements. Therefore, precision will tell us the percentage of predicted positive price movements that actually turned out to be correct, while recall will specify the total percentage of positive price movements that were classified as such.

4.1.2 Financial Evaluation

Given the nature of this work, it is necessary to consider a few other metrics that evaluate how profitable our trading strategies are. The financial metrics considered in this work follow:

- Return on Investment (ROI) - Ratio between net income and initial investment. The formula follows:

$$\text{ROI} = \frac{\text{NetProfit}}{\text{InitialInvestment}} \tag{4.1}$$

- Hit-rate - The ratio of winning or profitable trades over a period of time for a trading strategy:

$$\text{Hit-Rate} = \frac{\text{Profitable trades}}{\text{Total trades}} \tag{4.2}$$

The 0.1% trading fee was not considered when computing this metric;

- Sharpe Ratio - Introduced by William F. Sharpe in 1994 [40], it measures how much excess return is gained in comparison to a risk-free asset, while adjusting for the additional risk. The formula follows:

$$\text{Sharpe Ratio}(x) = \frac{\overline{R_x} - R_f}{StandardDeviation(R_x)} \tag{4.3}$$

where R_x is the series containing the returns of an investment x, R_f is the average return of a risk-free asset and the denominator is the Standard Deviation of investment r_x. The risk-free asset considered in this work was the U.S. Government Treasury bills, as it is the most commonly used in this formula. Lastly, the returns were measured with a time interval of 30 days (starting from May 18th, 2021).

- Sortino Ratio - Introduced by Frank A. Sortino in 1994, in his article entitled "Performance Measurement in a Downside Risk Framework" [41], the Sortino ratio is a variation of the Sharpe Ratio that only considers negative volatility in the denominator of the formula. This is done to address a limitation from the definition of the Sharpe Ratio - positive volatility (e.g abnormal returns) is considered to be bad and lower this ratio. To solve this problem, the Standard Deviation presented in the previous formula is replaced by the Downside Deviation, defined by:

$$\text{DownsideDeviation } = \sqrt{\frac{\sum (R_i - MAR)^2 \quad \text{where } R_i < MAR}{n}} \tag{4.4}$$

where R is the series containing the returns of the investment being evaluated, and MAR is the minimum acceptable return, considered to be 0 in this work.
The Sortino Ratio is then defined by:

$$\text{Sortino Ratio}(x) = \frac{\overline{R_x} - R_f}{DownsideDeviation(R_x)} \tag{4.5}$$

- Total trades - Total trades performed, where one trade includes both entering and exiting the market;
- Average time in market - Average trade duration (time between buying and selling).

.

4.2 Results

As mentioned earlier, each strategy started with 1000 US dollars. The traditional Buy and Hold strategy was used as a baseline.

With each transaction, a fee of 0.1% was considered. This is the fee charged by Binance on a regular trading account. Although apparently small, this heavily penalised extremely active trading strategies. Table 4.3 summarizes the results of each strategy, with and without considering trading fees. One thing to note is that the difference in returns with and without considering trading fees comes not only from the money lost on the fee itself, but also the loss of potential returns generated from the money lost.

Figure 4.1 shows the portfolio value of each strategy throughout the simulation, considering the 0.1% fee. XGBoost had by far the best performance, while the Buy and Hold strategy used as a baseline was the worst performer. Contrarily to what was expected, the ensemble voting strategy was outperformed by individual models (both tree-based models). A more in-depth analysis will be given throughout the following Sections.

Strategy	Final money (0.1% trading fee)	Final money (no fees)
Buy and Hold	646.53	647.83
Logistic Regression	1014.70	1102.56
Support Vector Machine	981.84	1041.54
Random Forests	1314.12	1516.25
XGBoost	1947.84	2297.46
Ensemble Voting	1212.19	1394.35

Table 4.3: Trading strategies' final portfolio value comparison.

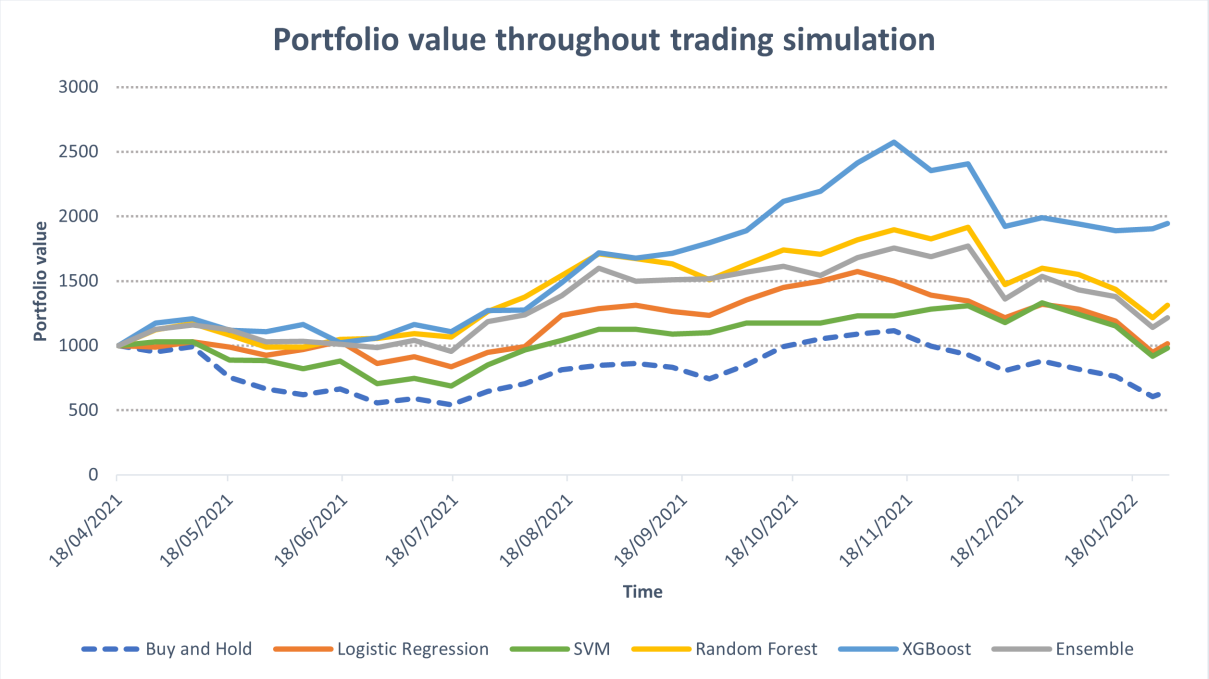

Figure 4.1: Portfolio values comparison throughout the simulation period.

4.2.1 Model evaluation

Table 4.4 summarizes the results obtained by the various models, as evaluated by machine learning metrics.

XGBoost had the best accuracy out of all models with 58.28%, beating the next highest by more than 3%. The other tree-based model - Random Forest, and the Ensemble voting model followed at 54.7% accuracy. The Logistic Regression and Support Vector Machine had significantly worse results in terms of accuracy with 52.8% and 52.2% respectively.

Another thing to note right away, is that all models except Random forests had extremely high Recall. This tells us that the models were mostly predicting positive moves. This is especially true when looking at the two worse performing models - LR and SVM.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Logistic Regression	52.8	51.4	86.4	64.4
Support Vector Machine	52.2	51.0	78.7	61.9
Random Forests	54.7	57.4	32.2	41.3
XGBoost	58.28	56.9	63.8	60.2
Ensemble Voting	54.7	53.2	70.3	60.6

Table 4.4: Model predictions evaluation.

4.2.1.A Features

Figure 4.2 shows the feature importance plot from the best performing model (XGBoost), as measured by ”weight” - the number of times a feature appears in a tree. From this graph, we can see that the most used feature is simply the percentage variation from the last candle.

These features were computed from the base features described in Table 3.1. There are a few prefixes and suffixes in this graph that need some explaining:

- Suffix ”(t-1)” identifies lagged features;
- Prefix ”Change_i” identifies features that represent percentage variations, where i represents the utilized time window;
- Prefix ”rolling_i” identifies a rolling average, where i represents the utilized time window;
- Suffix ”by_time” means the features were divided by Delta;
- Suffix ”difference” identifies features based on simple subtractions between the current and the last candle.

When no base feature name is present, then the based feature used was the Closing price. For example, ”change_3” is obtained by subtracting the Closing price (t-3) from the current Closing price and dividing the result by Closing price (t-3). In another example, ”rolling_20_difference” is the difference between the 20-candle Closing price rolling average computed in the current candle and the one computed on the previous candle.

The graph only shows the top 15 most important features, but more were passed on to our models, all based on similar operations performed on the base features presented in Table 3.1.

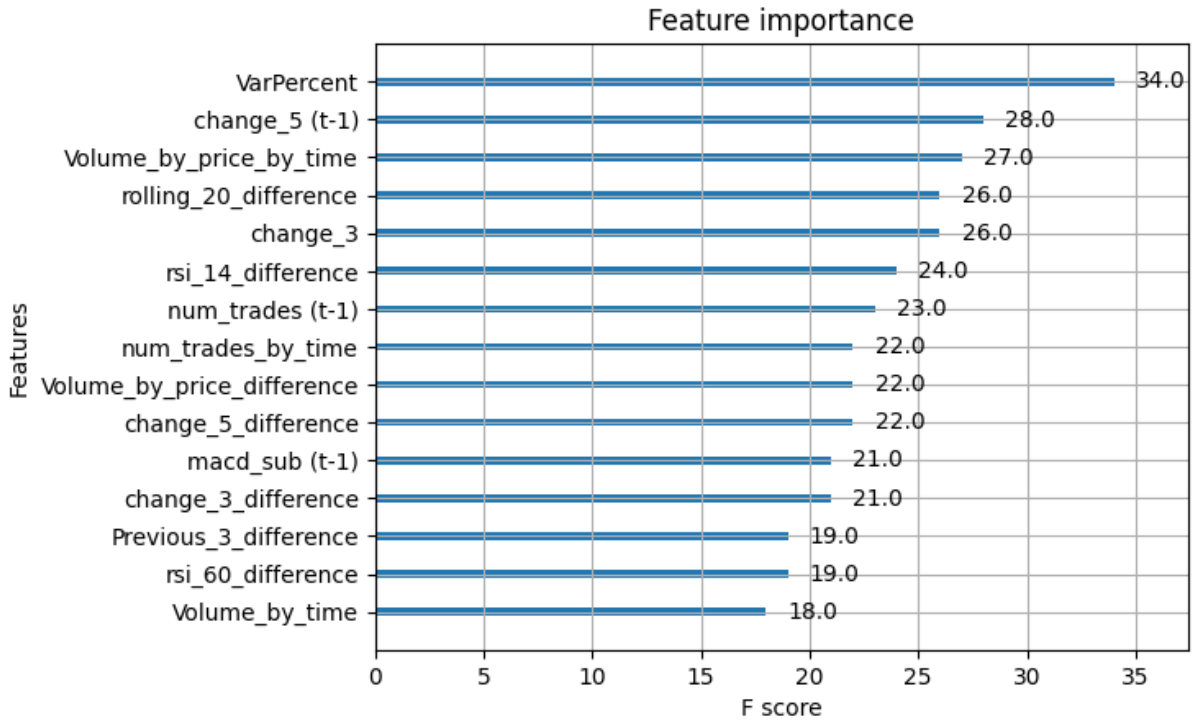

Figure 4.2: Feature importance plot

4.2.2 Financial evaluation

Table 4.5 shows that every model beat the Buy and Hold strategy that lost close to 35% of the initial investment during the 284 days included in the test set. The best performer, by far, was the XGBoost model with an ROI of 94.78%. This means that an initial investment of $1000 would yield $1947 if the XGBoost trading strategy was followed, while the same investment would yield a final portfolio value of $646.53 if the Buy and Hold strategy was followed. The XGBoost strategy was also the most active on the market with 83 total trades and an average Time in Market (TIM) per trade of 2 days and 6 hours.

The unbalanced predictions made by the Logistic Regression and the Support Vector Machine led to a long Average Time in Market. This means these strategies had higher exposure to market volatility, but

Trading Strategy	ROI	Hit-Rate	Sharpe Ratio	Sortino Ratio	# Trades	Average TiM
Buy & Hold	-35,35%	0	-0,150	-0,438	1	284 days
Logistic Regression	+1.47%	0.55	0,025	0,055	42	5.62 days
SVM	-1.82%	0.67	0,08	0,019	30	5.83 days
Random Forests	+31.41%	0.60	0,140	0,251	72	3 days
XGBoost	+94.78%	0.57	0,337	0,583	83	2,25 days
Ensemble Voting	+21.82	0.57	0,116	0,280	68	3 days

Table 4.5: Financial evaluation of each trading strategy.

still managed to beat the Buy and Hold strategy by a very significant margin.

Unfortunately, the hit-rate metric does not hold much value when analyzing these two strategies as longer trade times lead to fewer trades and higher volatility per trade, which means that each trade may have significantly different absolute returns. This explains why these two strategies have Hit-Rates comparable to the tree-based models but under-performed heavily in terms of the most important metric - Return on Investment.

Regarding the Sharpe Ratio, a score over 1 is usually considered good, while a score under 1 is considered bad. Unfortunately, despite our trading strategies heavily over-performing the market itself, all of them had what is commonly considered a bad result for this metric. The best one was XGBoost with a Sharpe Ratio of 0,337. As mentioned previously, the Sharpe ratio also penalizes high positive volatility (e.g. returns larger than usual), which is a factor that worsened the tree-based trading strategies' results.

On the other hand, the Sortino ratio only penalizes negative returns, and a strategy is said to have a good Sortino ratio if it stays above 2. The XGBoost strategy had the best Sortino Ratio with 0.583, so we can conclude again that, despite the abnormal returns generated by a few of our strategies, all of them scored negatively in this risk-adjusted ratio. Here, the main factor was the substantial price drop from November to December 2021, where XGBoost's strategy portfolio value dropped from around 2500 to 2000 - a 20% drop over 30 days. All the other strategies were also negatively affected by this drop.

One could argue that these baseline values for the Sortino and Sharpe ratios were created for inherently safer investments. Therefore, a better comparison to evaluate the risk-adjusted profitability of our trading strategies would be to compare them to the results obtained by other trading strategies operating in the same environment - the cryptocurrency market, during a similar time period, and using monthly returns in their Sharpe and Sortino ratios calculation.

4.2.2.A Entry graph analysis

Figures 4.3 and 4.4 show a graph comparing bitcoin's price and portfolio value using XGBoost's and the Ensemble's trading strategy respectively, throughout the 9-month test period. Before diving further into

the analysis, an important thing to notice in this graph is that the two lines correspond to different scales represented on the left and right Y axes. Although the percentage variations look similar at first sight, at their highest peaks, bitcoin's price change was around 26%, while XGBoost's peak ROI was around 150%. The graphs were intentionally designed this way to make bitcoin's smaller price fluctuations, from which our model profits, more evident.

Having this in mind, we can now analyze the graph more thoroughly. Right away, it is obvious that the XGBoost trading strategy clearly outperformed the Buy and Hold strategy, which would follow bitcoin's price fluctuations. While the former only went below the break-even line once, and only for brief moments, the latter stayed mostly under the break-even line. Also, while a Buy and Hold strategy would yield a negative final ROI of -35.35%, XGBoost's trading strategy would yield a final ROI of +94.78%.

It is also possible to confirm that, as intended, the percentage-based resampling led to a system that is much more active during high volatility periods - more than two-thirds of all trades were made in the first third of the simulation. On the other hand, the average time in market per trade was much smaller during the unstable times at the beginning of the simulation, which reduces the risk of trading during these highly volatile periods.

The percentage-based resampling also created an inherent stop-loss option for our model, as, after every negative 3.84% candle, a new decision point is created. This is another reason why this model responds well to high volatility periods.

The trading strategy dealt better with sharp drops like the one that took place at the end of May 2021, rather than longer bear markets like the one observed between November 2021 and February 2022. One possible explanation might be that this kind of sharp drops lead to predictable rebounds, while slower drops do not. This is something that could be explored in future work, including what type of features could help a model with percentage-based candles to better predict longer bear markets.

As for the ensemble, the majority vote strategy led to a system that took too long to react to drops. This effect and the optimal ensemble strategy will be further studied in Section 4.3.1.

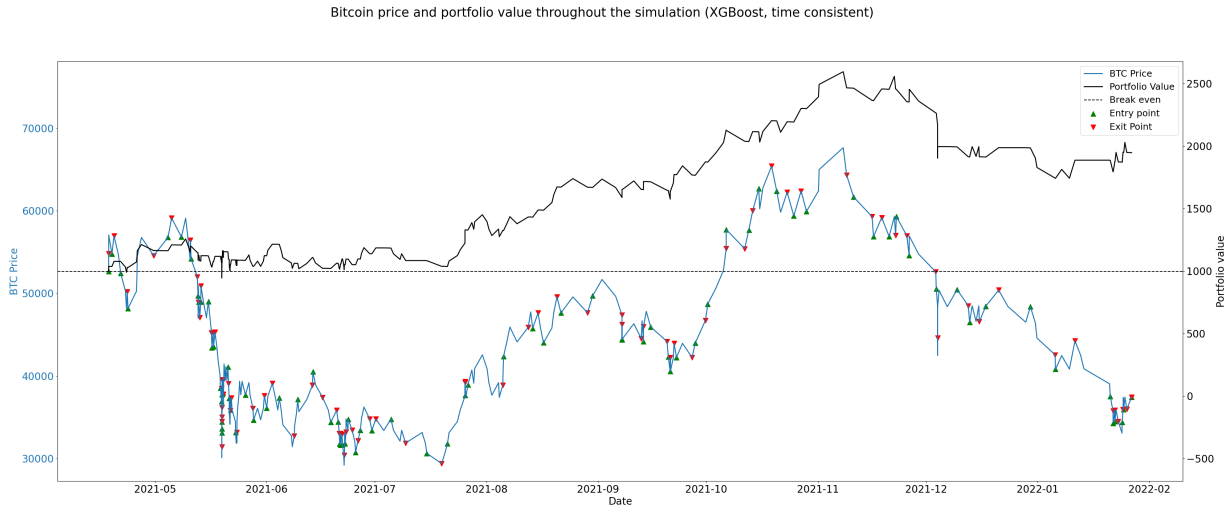

Figure 4.3: XGBoost trading strategy entry graph.

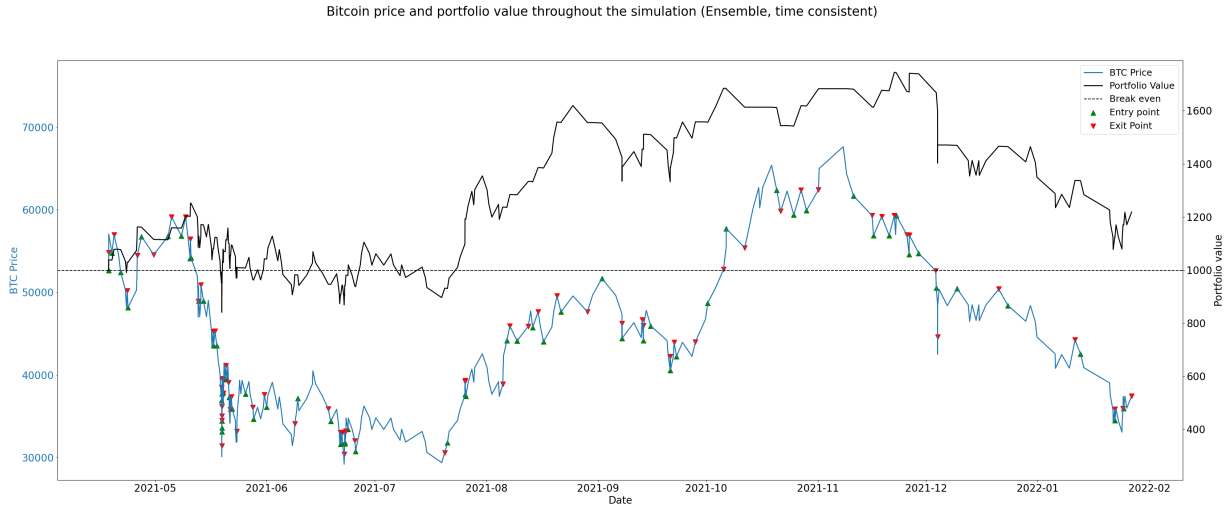

Figure 4.4: Three-model ensemble (majority vote) trading strategy entry graph.

4.3 Case Studies

In this Section, three case studies are presented where a specific parameter or approach is more thoroughly analyzed, by experimenting with different values or strategies. A short explanation of each case study follows:

- Optimal ensemble voting thresholds: In the first study, various strategies to combine the single-model predictions are analyzed.
- Asymmetric resampling thresholds: A comparative analysis between the results obtained using

asymmetric candles, and results obtained with symmetric ones.

- Percentage resampling threshold value: A range of values for the resampling threshold are tested and the optimal value is determined.

The following subsections go over each of these case studies more thoroughly.

4.3.1 Optimal ensemble voting thresholds

This case study intends to find the most optimal way to combine the single-model predictions, in order to obtain the most profitable system. Two of these strategies obtained results that beat the XGBoost trading strategy.

Different buying and selling thresholds were tested, and also different-sized ensembles. For the smaller-sized ensembles, the models were picked based on their individual results. Thus, the three-model ensemble consisted of the Random Forest, XGBoost, and SVM models, while the two-model ensemble consisted only of the tree-based models.

One could argue that, to design a better-performing ensemble model, correlation tests should have been performed between each individual model's predictions, as the lowest correlated models would, in theory, provide the best results. However, due to the very large performance difference between all models, this did not prove true.

Having different buying and selling thresholds means that, at each decision point (every time a candle closes), the trading system's control function *f* adopted the following behavior:

$$f = \begin{cases} "BUY" & \text{if not } currently\_in() \text{ and } up\_vote\_sum \geq tr\_in \\ "SELL" & \text{if } currently\_in() \text{ and } down\_vote\_sum \geq tr\_out \\ \text{"Do nothing"} & else \end{cases}$$

where *currently_in* is a function that returns True, if the system currently holds bitcoin, or False if not, *up_vote_sum* (*down_vote_sum*) is equal to the sum of models that predicted the market would go up (down) in the next candle, and *tr_in* and *tr_out* are the predefined buying and selling thresholds respectively.

The original single-model predictions, computed with the models described previously, were used in this experience. For the trading simulation, an initial portfolio value of $1000 was considered. Tables 4.6, 4.7, and 4.8 summarize the obtained results for ensembles containing four, three, and two models respectively.

Analyzing the results, one can observe that independently of the number of models utilized in the ensemble, the best result always pertained to the experiences that had simultaneously the highest possible "Buy" threshold and the lowest possible "Sell" threshold.

(Sell thr. / Buy thr.)	Final money	# Trades	Average TiM	ROI	Hit-Rate
(2 / 2)	1612,11	83	2 days 12:34:09	61,21%	0,61
(2 / 3)	1201,63	71	2 days 17:36:50	20,16%	0,58
(2 / 4)	1792,99	56	2 days 10:18:11	79,3%	0,59
(3 / 2)	1213,51	43	5 days 12:21:41	21,35%	0,6
(3 / 3)	1086,63	38	5 days 19:13:25	8,66%	0,61
(3 / 4)	1410,81	30	5 days 18:34:40	41,08%	0,7
(4 / 2)	944,66	12	22 days 23:47:15	-5,53%	0,5
(4 / 3)	872,8	11	24 days 16:00:05	-12,72%	0,45
(4 / 4)	915,85	9	28 days 12:58:53	-8,41%	0,44

Table 4.6: Four-model ensemble simulation results, given different Buy and Sell thresholds.

(Sell thr. / Buy thr.)	Final money	# Trades	Average TiM	ROI	Hit-Rate
(1 / 2)	1630,14	103	1 days 13:39:49	63,01%	0,6
(1 / 3)	2197,6	75	1 days 10:41:52	119,76%	0,64
(2 / 2)	1218,18	68	2 days 23:53:05	21,82%	0,57
(2 / 3)	1878,98	53	2 days 15:47:54	87,9%	0,58
(3 / 2)	1139,62	24	10 days 19:22:32	13,96%	0,58
(3 / 3)	1119,11	19	11 days 01:11:09	11,91%	0,63

Table 4.7: Three-model ensemble simulation results, given different Buy and Sell thresholds.

(Sell thr. / Buy thr.)	Final money	# Trades	Average TiM	ROI	Hit-Rate
(1 / 1)	1696,39	108	1 days 18:31:13	69,64%	0,59
(1 / 2)	2070,38	85	1 days 18:05:48	107,04%	0,61
(2 / 1)	1266,38	58	4 days 09:24:41	26,64%	0,53
(2 / 2)	1561,61	52	4 days 07:52:32	56,16%	0,5

Table 4.8: Two-model ensemble simulation results, given different Buy and Sell thresholds.

Generally speaking, the final portfolio value increased with higher ”Buy” thresholds and with lower ”Sell” thresholds. Both of these fluctuations lead to a smaller average time in market, which may have led to a more risk-averse trading system. Also, in machine learning terms, having more models agreeing before entering the market would also increase our ”Buy moment” precision, while allowing for fewer models to sell our position increased our ”Sell moment” recall. This translates to higher Hit-Rates when looking at experiments with large ”Buy” thresholds and small ”Sell” thresholds.

Comparing these ensemble models to the previously mentioned majority vote ensemble (the (2 / 2) three-model ensemble in this experience), we can conclude that a few other ensemble strategies greatly outperformed it. More specifically, some ensemble strategies with large ”Buy” thresholds and small ”Sell” thresholds more than tripled the previously mentioned majority voting ensemble’s Return on Investment.

Despite XGBoost having abnormal returns when compared to the other members of the ensemble, it was still outperformed by two of the ensemble strategies. This goes in accordance with the findings of

multiple papers ([14], [34], [35]) where ensembles outperformed their constituents.

The best overall result was obtained with the three-model ensemble, with a buying threshold of 3 and a selling threshold of 1. It achieved an ROI of 119,76%, and made 75 trades with 0,6 hit-rate (45 winning trades), during the simulation period of 284 days. It had the lowest average time in market per trade, with a value inferior to 1 day and 11 hours.

Figures 4.5 and 4.6 show the entry graphs for the 3-model ensemble, with a buying threshold of 3 and a selling threshold of 1, and the 2-model ensemble, with a buying threshold of 2 and a selling threshold of 1, respectively. Comparing them to Figure 4.4, which represents the basic majority-vote ensemble, we can confirm that the larger buying thresholds led to a higher percentage of successful entries while the lower selling thresholds made the system more reactive to price drops.

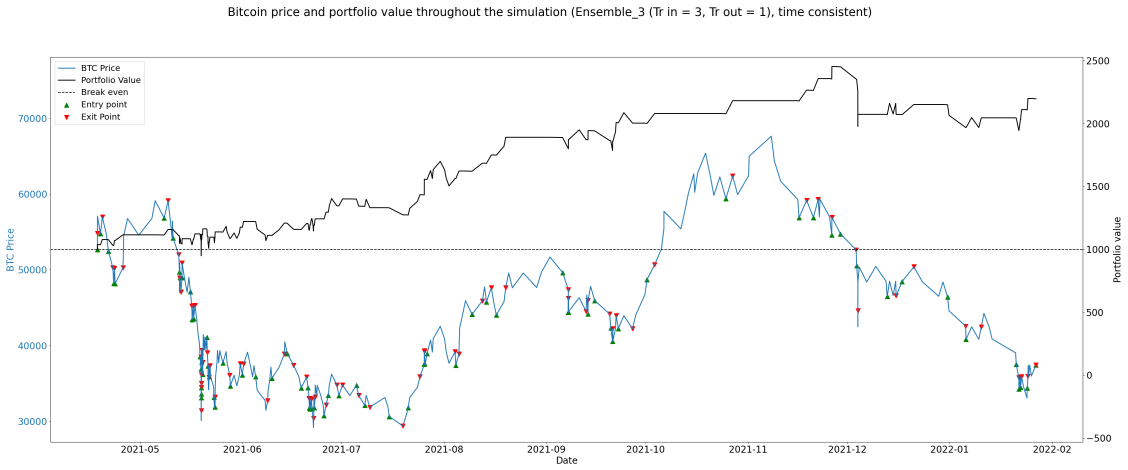

Figure 4.5: Three-model ensemble trading strategy entry graph. Buying threshold = 3, Selling threshold = 1

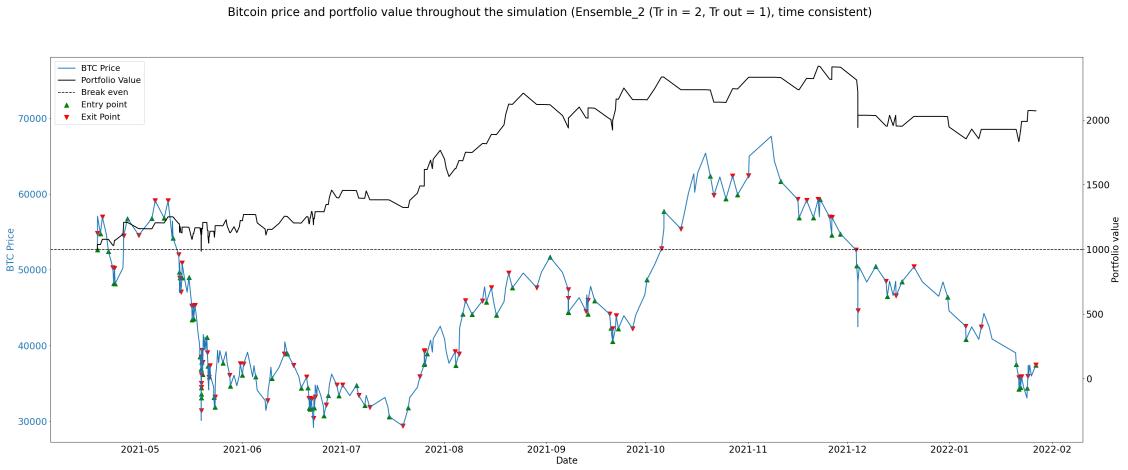

Figure 4.6: Two-model ensemble trading strategy entry graph. Buying threshold = 2, Selling threshold = 1

4.3.2 Asymmetric resampling thresholds

As mentioned before, having asymmetric candles that lead to break-even positions (not considering fees) proved to be essential in getting our trading system to be more profitable.

In this subsection, a comparison between the originally presented system with asymmetric candles, and a system with symmetric candles is provided. The chosen positive candle size for this experience was 4%. This means that, for both systems, the positive threshold is equal to 4%, but, while the negative threshold used by the symmetric candle system is -4%, the one used by the original system is -3.84%.

Every model was retrained with the new candles, then, new predictions were made and a new trading simulation was performed. The hyper-parameters were slightly adjusted for the new system, as the total number of generated data points was smaller.

The last thing to note is that, because the final generated candles are slightly different, so is the simulation period starting date: while the original system starts trading on 2021-04-18, the symmetric candles system's simulation period started on 2021-04-12. This is due to the fact that the candles of each system do not line up perfectly in time, however, bitcoin's starting price was similar for both systems. Also, for both systems, the trading simulation period is still composed only of data that was not present in the respective training phases.

The experiment results are detailed in Table 4.9. Figures 4.7 and 4.8 represent the Random Forest strategy's entry graph, with asymmetrical and symmetrical candles, respectively. These graphs were chosen for comparison because the RF model achieved similar accuracy with both candle types.

Model	Candles	Accuracy	Hit-Rate	ROI	Final Money
Logistic regression	Asymmetric	52,87%	0,55	1,47%	1014,7
	Symmetric	52,96%	0,51	-13,80%	862,04
SVM	Asymmetric	52,23%	0,67	-1,82%	981,84
	Symmetric	52,63%	0,55	-17,46%	825,39
Random Forest	Asymmetric	55,41%	0,6	31,41%	1314,12
	Symmetric	55,26%	0,53	12,86%	1128,6
XGBoost	Asymmetric	58,28%	0,57	94,78%	1947,84
	Symmetric	54,93%	0,49	15,27%	1152,7
Ensemble	Asymmetric	54,78%	0,57	21,82%	1218,19
	Symmetric	53,29%	0,49	-12,57%	874,33

Table 4.9: Asymmetric vs Symmetric candles comparison.

Analyzing the results, we can conclude that every asymmetric model beat its symmetric counterpart in Hit-Rate and, most importantly, in Return on Investment.

In terms of accuracy, it was mostly balanced between both model versions, with the exception of XGBoost which had a larger difference between both versions. We can also observe that, even when the symmetric models beat their counterparts in accuracy, they still lost in Hit-Rate and ROI.

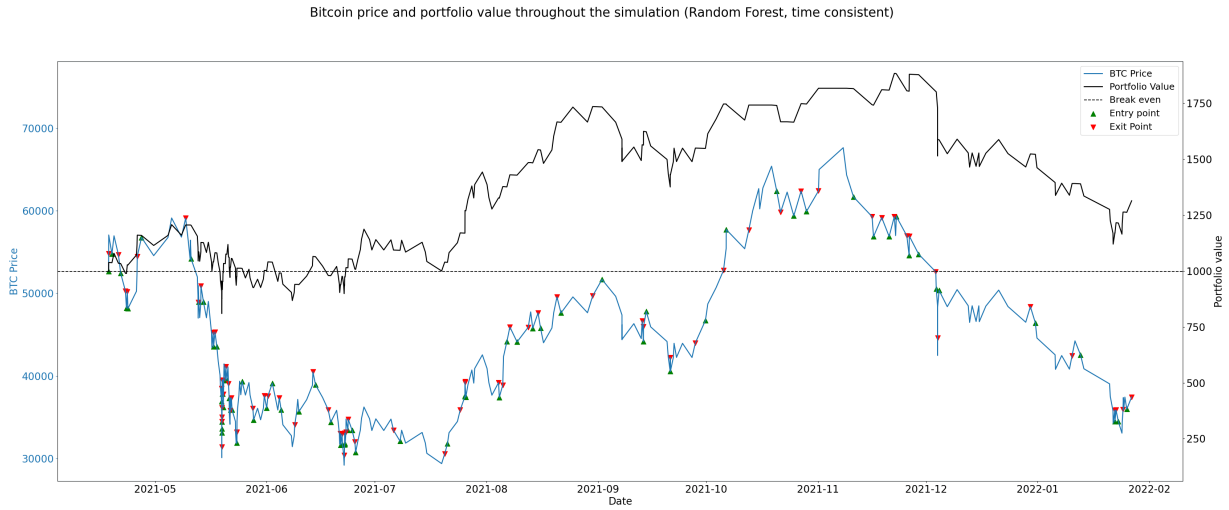

Figure 4.7: Asymmetrical candles Random Forest entry graph.

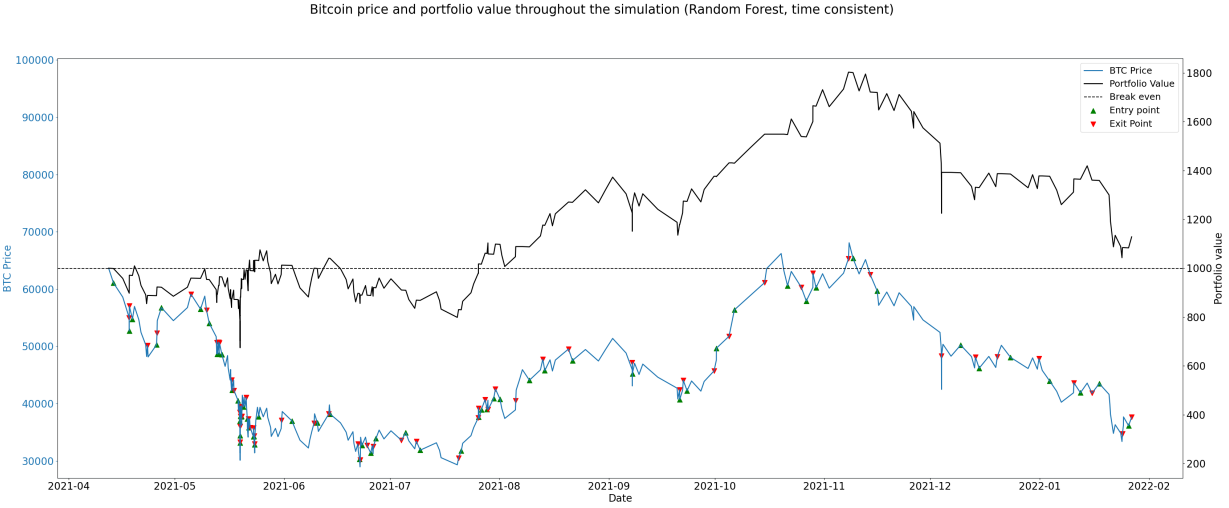

Figure 4.8: Symmetrical candles Random Forest entry graph.

So it would be natural to conclude that asymmetrical candles were the better choice. However, one can argue that the success factor was that the negative candle threshold was larger (closer to 0) and that the results could be even better if the negative threshold was even closer to 0 (more asymmetrical). This would lower our inherent stop-loss, by creating more decision points during a market drop and could potentially improve performance.

On the other hand, having thresholds that lead exactly to break-even points could also be the main success factor, as it better lines up with the human way of looking at this task by creating decision points at similar price points: if bitcoin's price falls after a decision point, and then rises again, a new decision

point will be created at exactly the same price point. This is a topic that could be further explored in future work.

4.3.3 Percentage resampling threshold

Choosing the optimal percentage threshold proved to be an essential step to having a successful trading strategy. Setting this threshold too low will lead to losing money due to fees, while setting this threshold too high leads to very inactive strategies that can stay both in or out of the market for really long periods of time.

For this experience, only the previously described XGBoost model was analyzed. As with the original system, asymmetric candles were used and the break-even negative threshold that corresponds to each positive threshold was calculated through expression 3.1.

Various thresholds were tested and the results are summarized in Table 4.10 and Figure 4.9. As different thresholds lead to drastically different amounts of training data, a round of hyper-parameter optimization was performed before each experience.

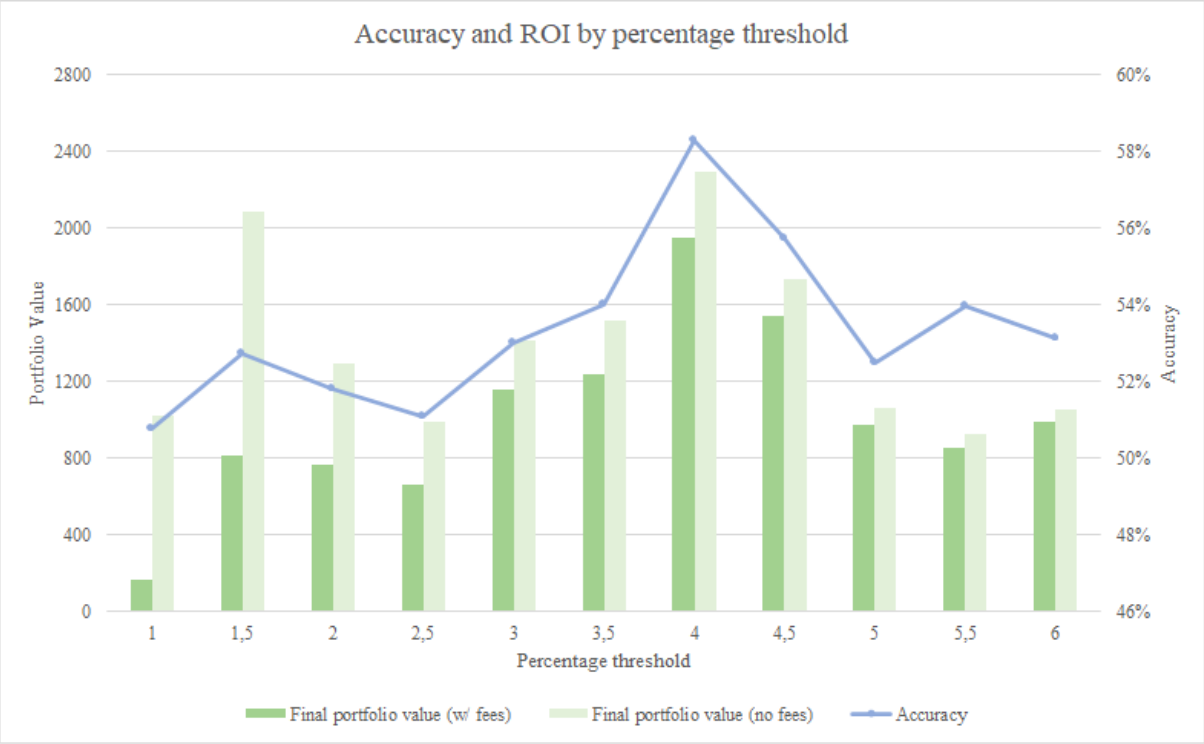

Figure 4.9: Accuracy and ROI by resampling threshold

We can conclude that the model performed better for the thresholds of 4 and 4.5%, where the accuracy stayed above 55%. This does not necessarily mean that these thresholds lead to the most predictable candles, as the amount of training data for the larger thresholds was quite lower. This is a

Threshold	Accuracy	Final value (w/ fees)	Final value (no fees)	Total trades	Avg TiM per trade
1	50,78%	161,3	1024,73	924	0 days 03:59:38
1,5	52,74%	812,73	2083,63	471	0 days 07:26:18
2	51,83%	768,9	1293,64	260	0 days 15:19:58
2,5	51,10%	658,48	986,47	202	0 days 17:21:00
3	53,03%	1158,96	1411,46	99	2 days 06:41:46
3,5	54,00%	1238,67	1514,58	101	1 day 22:10:04
4	58,28%	1947,84	2297,46	83	2 days 06:39:27
4,5	55,73%	1540,9	1732,26	59	2 days 17:46:46
5	52,48%	973,93	1060,38	43	4 days 13:58:29
5,5	53,98%	855,37	927,57	41	4 days 15:06:42
6	53,15%	988,52	1057,06	34	3 days 09:49:58

Table 4.10: XGBoost results with different sized candles.

topic that could be further explored in future work, by using a larger training period pulled from a different data source (Binance only has historic data starting from 2018). Despite this, we can make the more conservative conclusion that, using this set of features, medium-range (3 to 4.5%) price moves were more predictable than the smaller (1 to 2.5%) price changes. As expected, the number of trades was inversely proportional to the chosen resampling threshold. Also, larger number of trades resulted in smaller average time on market per trade which was also expected. In other words, smaller thresholds resulted in more active trading strategies.

As for the profitability analysis, we can see that thresholds under 3% are not profitable due to a significant amount of money being lost on fees. More specifically, the threshold of 1.5% was a small outlier in terms of accuracy. This small boost in accuracy with a low threshold led to abnormal returns when disregarding trading fees, but still represents a loss when considering them.

On the other side of the spectrum, thresholds above 4.5% were not active enough and were therefore too exposed to the market volatility, leading to portfolio depreciation despite the positive prediction accuracy.

4.4 Overall Analysis

In this Section, we summarize the most important results obtained in this work and make a few overall comparisons regarding the different presented approaches.

The best overall result was obtained by an Ensemble Voting strategy, composed of three models with a Buy threshold of 3 and a Sell threshold of 1 with an ROI of 119.76% over 284 days. The best single model strategy was the one based on XGBoost's predictions, achieving an ROI of 94.78% over the same period. These are really promising results when compared to the baseline Buy and Hold strategy that

yielded a negative ROI of -35.35% over the same period.

As for the case studies, each of them had an important contribution:

1. **Optimal ensemble voting thresholds** - This case study found the optimal way of combining our single model predictions. Furthermore, it proved that ensembles with larger Buy thresholds and smaller Sell thresholds were more successful than other ensemble strategies.

2. **Asymmetric resampling thresholds** - This case study proved that the asymmetric candles obtained from equation 3.1 over-performed the symmetric candles, for every model tested.

3. **Percentage resampling threshold** - This case study confirmed that the optimal resampling threshold (candle size) was around 4%, as chosen in our proposed solution. Furthermore, it analyzed the effects of lowering and increasing this threshold.

5

Conclusion

Contents

5.1 Summary and Achievements

In this work, a solution that massively outperformed the market was proposed. Its design combined the findings of multiple state-of-the-art systems and introduced a new resampling approach that better lines up the target variable with the goal of being profitable.

Four machine learning models were trained with 80% of the available data and tested in the following 20%. A fifth model - Ensemble Voting - was derived from the predictions of the previous models. All our models attained accuracy scores over 50%, thus beating a baseline random model. The XGBoost model was the clear winner with 58.28% accuracy, and the other tree-based method - Random Forest - also had a respectable result with 54%.

Multiple trading strategies were derived from the model predictions and tested in the 9-month simulation period (or test set), from April 2021 to January 2022. In this period, while the market dropped by more than 35%, our single model strategies' ROI ranged from -1.82% to +94.78%. Again, the tree-based models generated the winning trading strategies, being both more active and more profitable.

In one of our case studies, the best way to combine the single-model predictions was investigated, and two strategies outperformed the best single-model trading strategy with ROIs of 107.04% and 119.76%. From this study, we concluded that strategies with high "Buy" thresholds and low "Sell" thresholds were consistently more profitable.

These results are very promising, especially given the very turbulent times in which the system was tested that included both a bull and a bear market. Despite this, a few opportunities for improvement still exist, and these will be detailed in the following subsection.

5.2 Future Work

In this subsection, some of our system's limitations and their possible solutions are addressed, along with suggestions that may lead to performance improvements in future work.

- In this work, when our system received either a "Buy" or "Sell" signal, all of its portfolio value was changed from one asset to the other. Instead, one could invest only a percentage of its portfolio at a time. This percentage could be fixed or defined by a given indicator or even by the prediction probability, outputted by the model.
- To achieve better "Buy" signal precision, one could set the positive prediction probability threshold to a number larger than the default value of 0.5. Different thresholds could be set for entering or leaving the market, similar to what was done in this work with the different ensemble voting thresholds. Although this would make the system less active, it would improve Hit-Rate. In turn,

this would possibly allow the user to lower the resampling threshold (candle size), making the system more active (or as active as before).

- Regarding the features used, a plethora of different features could be used. One could use different technical indicators, blockchain-based features, or even social networks sentiment analysis.
- Only long positions were considered in this work. One could potentially improve the results by also considering short positions.

Bibliography

[1] Nakamoto, Satoshi. ”Bitcoin: A peer-to-peer electronic cash system.” Decentralized Business Review (2008): 21260.

[2] Hileman, Garrick, and Michel Rauchs. ”Global cryptocurrency benchmarking study.” Cambridge Centre for Alternative Finance 33 (2017): 33-113.

[3] Forbes article, https://www.forbes.com/sites/lawrencewintermeyer/2021/08/12/institutional-money-is-pouring-into-the-crypto-market-and-its-only-going-to-grow/?sh=6fb127ad1459. Last accessed 6 Jan 2022

[4] Binance homepage, www.binance.com. Last accessed 23 Nov 2021

[5] Coinmarketcap - Market capitalization evolution, https://coinmarketcap.com/charts/. Last accessed 23 Nov 2021

[6] Foley, Sean, Jonathan R. Karlsen, and Tālis J. Putniņš. ”Sex, drugs, and bitcoin: How much illegal activity is financed through cryptocurrencies?.” The Review of Financial Studies 32.5 (2019): 1798-1853.

[7] Oxford Languages, https://languages.oup.com/google-dictionary-en/. Last accessed 23 Nov 2021

[8] Fortune article, https://fortune.com/2021/04/25/bitcoin-btc-value-drop-use-case-currency-digital-gold/. Last accessed Jan 6 2022

[9] Coinmarketcap - Listed cryptocurrencies, https://coinmarketcap.com/. Last accessed Jan 6 2022

[10] Adhikari, Ratnadip, and Ramesh K. Agrawal. ”An introductory study on time series modeling and forecasting.” arXiv preprint arXiv:1302.6613 (2013).

[11] Abu-Mostafa, Yaser S., and Amir F. Atiya. ”Introduction to financial forecasting.” Applied intelligence 6.3 (1996): 205-213.

[12] Dettori, Joseph R., and Daniel C. Norvell. "The anatomy of data." Global spine journal 8.3 (2018): 311-313.

[13] Machine Learning Mastery - Categorical data encoding, https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/. Last accessed Nov 22 2021

[14] Borges, Tome Almeida, and Rui Ferreira Neves. "Ensemble of machine learning algorithms for cryptocurrency investment with different data resampling methods." Applied Soft Computing 90 (2020): 106187.

[15] SVM example, https://en.wikipedia.org/wiki/Support-vector_machine Last accessed Jan 5 2022

[16] KNN example, https://scikit-learn.org/0.22/auto_examples/neighbors/plot_classification.html. Last accessed Jan 5 2022

[17] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[18] Kirkpatrick II, Charles D., and Julie A. Dahlquist. Technical analysis: the complete resource for financial market technicians. FT press, 2010.

[19] Iansiti M and Lakhani K R 2017 The Truth About Blockchain Harvard Business Review, Harvard University, hbr.org/2017/01/the-truth-about-blockchain. Last accessed Nov 22 2021.

[20] LeMahieu, Colin. "Nano: A feeless distributed cryptocurrency network." Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018) 16 (2018): 17.

[21] Fama, Eugene F. "Efficient capital markets a review of theory and empirical work." The Fama Portfolio (2021): 76-121.

[22] Eugene F. Fama. "Efficient Capital Markets: II." The Journal of Finance, vol. 46, no. 5, [American Finance Association, Wiley], 1991, pp. 1575–617, https://doi.org/10.2307/2328565.

[23] Urquhart, Andrew. "The inefficiency of Bitcoin." Economics Letters 148 (2016): 80-82.

[24] Bariviera, Aurelio F. "The inefficiency of Bitcoin revisited: A dynamic approach." Economics Letters 161 (2017): 1-4.

[25] Le Tran, Vu, and Thomas Leirvik. "Efficiency in the markets of crypto-currencies." Finance Research Letters 35 (2020): 101382.

[26] Baur, Dirk G., Kihoon Hong, and Adrian D. Lee. "Bitcoin: Medium of exchange or speculative assets?." Journal of International Financial Markets, Institutions and Money 54 (2018): 177-189.

[27] Bouri, Elie, et al. ”On the hedge and safe haven properties of Bitcoin: Is it really more than a diversifier?.” Finance Research Letters 20 (2017): 192-198.

[28] Pyo, Sujin, et al. ”Predictability of machine learning techniques to forecast the trends of market index prices: Hypothesis testing for the Korean stock markets.” PloS one 12.11 (2017): e0188107.

[29] Jaquart, Patrick, David Dann, and Christof Weinhardt. ”Short-term bitcoin market prediction via machine learning.” The Journal of Finance and Data Science 7 (2021): 45-66.

[30] Ji, Suhwan, Jongmin Kim, and Hyeonseung Im. ”A comparative study of bitcoin price prediction using deep learning.” Mathematics 7.10 (2019): 898.

[31] Kwon, Do-Hyung, et al. ”Time series classification of cryptocurrency price trend based on a recurrent LSTM neural network.” Journal of Information Processing Systems 15.3 (2019): 694-706.

[32] Alessandretti, Laura, et al. ”Anticipating cryptocurrency prices using machine learning.” Complexity 2018 (2018).

[33] de Souza, Matheus José Silva, et al. ”Can artificial intelligence enhance the Bitcoin bonanza.” The Journal of Finance and Data Science 5.2 (2019): 83-98.

[34] Sebastião, Helder, and Pedro Godinho. ”Forecasting and trading cryptocurrencies with machine learning under changing market conditions.” Financial Innovation 7.1 (2021): 1-30.

[35] Mallqui, Dennys CA, and Ricardo AS Fernandes. ”Predicting the direction, maximum, minimum and closing prices of daily Bitcoin exchange rate using machine learning techniques.” Applied Soft Computing 75 (2019): 596-606.

[36] Nevasalmi, Lauri. ”Forecasting multinomial stock returns using machine learning methods.” The Journal of Finance and Data Science 6 (2020): 86-106.

[37] Martin, James H. “Logistic Regression.” Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Pearson Prentice Hall, Upper Saddle River, NJ, 2009.

[38] Bishop, Christopher M. Pattern Recognition and Machine Learning. New York :Springer, 2006.

[39] Random forest explanation, https://towardsdatascience.com/understanding-random-forest-58381e0602d2. Last accessed Jan 13 2022

[40] Sharpe, William F. ”The sharpe ratio.” Streetwise–the Best of the Journal of Portfolio Management (1998): 169-185.

[41] Sortino, Frank A., and Lee N. Price. ”Performance measurement in a downside risk framework.” the Journal of Investing 3.3 (1994): 59-64.

[42] XGBoost documentation, https://xgboost.readthedocs.io/en/stable/tutorials/model.html. Last accessed Jan 5 2022