AALBORG UNIVERSITY
DENMARK

Mathematics-Economics – Financial Engineering
10th Semester Master's Thesis

On Machine Learning Based Cryptocurrency Trading

Authors:
Willam Geneser Bach
Kasper Lindblad Nielsen

Department of Mathematical Sciences
Mathematics-Economics
Skjernvej 4A
9220 Aalborg Øst
Phone +45 99 40 88 01
http://math.aau.dk

Title:
On Machine Learning Based Cryptocurrency Trading

Theme:
Specialization within Financial Engineering

Project period:
February 1st - June 7th, 2018

Project group:
MAOK10 5.219B

Members:
William Geneser Bach
Kasper Lindblad Nielsen

Supervisor:
Eduardo Vera-Valdés

Completed:
June 7th, 2018

Page Numbers: 121

Abstract:

In this thesis we examine the effectiveness of several machine learning algorithms for trading cryptocurrencies on Binance.
First we set up a trading framework, which allows us to test several parametrizations of the cryptocurrency trading data and examine which are best suited for the algorithms. Within this framework we aggregate data at several intervals, add multiple factors and incorporate technical analysis indicators to assist the models. We then classify historical data into buys or stays, and finally difference, lag, and split it. This framework enables us to set up a supervised classification problem that we solve by optimizing the data parametrizations and algorithm configurations.
We consider four algorithms: generalized linear models (logistic regression), neural networks, gradient boosting, and random forests, and briefly describe the theory relevant to understand these algorithms before proceeding to the task of applying them in the trading framework.
Towards the end of the thesis we test the optimized models on six trading pairs trading against Tether: Bitcoin, Ethereum, Binance Coin, NEO, Litecoin, and Bitcoin Cash.
We end the thesis by providing some concluding remarks and our thoughts on further developments to improve the framework.

Preface

This master's thesis is written in the spring of 2018, by group 5.219B from the Department of Mathematical Sciences at Aalborg University. The group consists of two 10th semester mathematics-economics students. We recommend reading the chapters in order, and the intended audience of this thesis are graduate students on a mathematical degree or individuals of similar comprehension level. In-text references are of the format: *author's last name* (*year of publication*), or (*author's last name*, *year of publication*, *page number(s)*) when referencing specific pages. Whenever an equation is referenced, we write (x) and this should be read as "equation x". When reading a plot we suggest first reading the caption, then the legend (top-right corner if applicable), and finally examine the plot itself.

All data used in the thesis is gathered from the cryptocurrency exchange Binance using their API, specifically the 'klines' endpoint. The paper is written in LaTeX, computations performed solely in R. A complete list of the R-packages used is found in Appendix B.1.

William Bach
`<wbach12@student.aau.dk>`

Kasper Lindblad Nielsen
`<kaniel12@student.aau.dk>`

Aalborg University, June 7th, 2018

Resume (Danish)

I dette speciale er hovedfokus på at opsætte en ramme for, hvordan man automatiseret kan handle kryptovalutaer på kryptobørsen Binance ved brug af maskinlærings algoritmer. Herunder er en beskrivelse af indholdet i hvert kapitel i specialet.

I Kapitel 1 giver vi først en kort introduction til, hvordan man handler kryptovaluta og dernæst opsætter vi en ramme for, hvordan vi vil lave automeret handel af kryptovalutaer, hvilket omfatter anskaffelsen og forberedelsen af data til maskinlærings algoritmerne. Vi henter data gennem Binances API og forbereder det ved at aggregere det til større intervaller, tilføje forskellige faktorer, klassificere det så vi har en respons vektor til algoritmerne, og deler det op i trænings, validerings og test sæt.

I Kapitel 2, 3 og 4 beskriver vi den nødvendige teori for at forstå brugen af de fire maskinlærings algoritmer, anvendt i dette speciale. Kapitel 2 handler om generaliserede lineære modeller, specifikt logistisk regression, Kapitel 3 om de to træbaserede modeller: gradient boosting og random forest, og Kapitel 4 om neurale netværk.

I Kapitel 5 bruger vi data fra IMDb filmanmeldelser for at vise eksempler på implementeringen af det neurale netværk og de to træ-baserede modeller.

I Kapitel 6 bruger vi de fire maskinlærings algoritmer til at klassificere data som køb eller ej på BTC-USDT parret. Vi tager udgangspunkt i en grådig-algoritme søgen for at finde de bedst egnede data parameteriseringer.

I Kapitel 7 forsøger vi at yderligere forbedre de data parameteriseringer vi fandt virkede bedst for modellerne på BTC-USDT parret. Da gradient boosting og random forest klarer sig betydeligt bedre end de to andre modeller fortsætter vi analysen med kun de to. Vi tester om en rullende prædiktion forbedrer modellerne, dernæst undersøger vi betydningen af størrelsen på trænings sættet.

I Kapitel 8 tester vi de forbedrede modeller på uset data for BTC-USDT parret og andre kryptovaluta par. Vi holder data parameteriseringen og model konfigurationen ens for alle parrene og ser en profit på fem af de seks par vi betragter.

I Kapitel 9 afrunder vi specialet med nogle konkluderende kommentarer på resultaterne. Denæst deler vi vores tanker omkring videre udvikling af den opsatte ramme til at basere handelsbeslutninger på.

Sidst i specialet er vores appendikser og bibliografiliste.

Contents

Part I

Framework

1 | Introduction

On January 1st, 2017 a total of 617 different cryptocurrencies were tracked on CoinMarketCap (2018) with an aggregated market cap of $17,700,314,429$ USD. A year later, on January 7th, 2018 the number of cryptocurrencies increased to 1355 with a market cap of $823,859,466,471$ USD, and by the time of starting this thesis there is 1483 cryptocurrencies with a market cap of $442,894,135,097$ USD. A visualization of the market cap in this period is shown in Figure 1.1, which illustrates growth and volatility of the cryptocurrency space. Cryptocurrencies and the associated blockchain technology is being widely adopted on a large scale with a multitude of large established companies adopting cryptocurrencies and blockchain technology. The central subject of this thesis

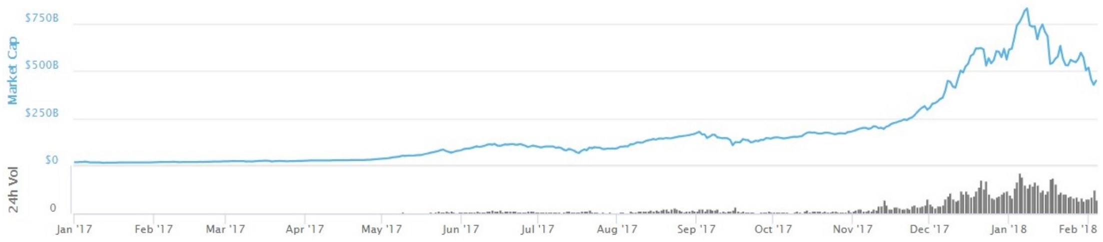


Figure 1.1: The aggregated cryptocurrency market cap in the period from January 1st, 2017 to February 4th, 2018, as reported by CoinMarketCap (2018).

is the trading of cryptocurrencies, specifically automated trading. Throughout this thesis we attempt to apply a selection of machine learning algorithms to cryptocurrency trading data with the objective of predicting profitable trades. In Section 1.1 we provide a brief introduction to the trading of cryptocurrencies. Section 1.2 presents the hypothesis that we are repeatedly testing throughout the thesis when trying to predict profitable trades. Sections 1.3-1.5 presents how we obtain and process cryptocurrency trading data in order to facilitate the prediction of trades.

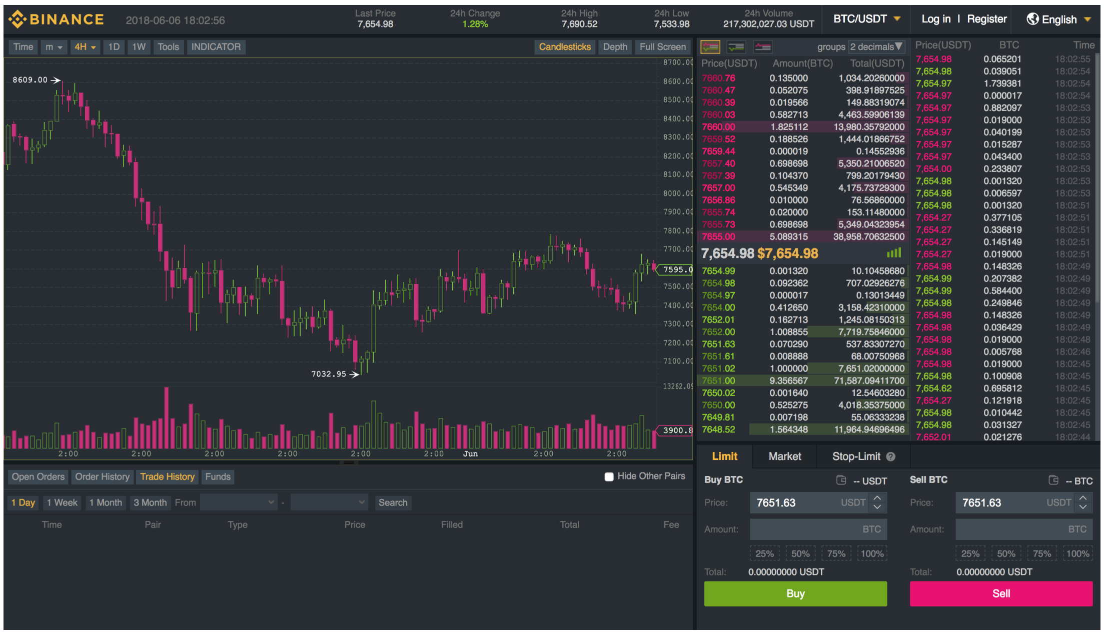
1.1 | Cryptocurrency Trading

Compared to traditional financial markets, the cryptocurrency market is highly unregulated in that it is, for the most part, decentralized and open-source, meaning anyone can create a cryptocurrency and anyone can trade them. Due to the financial regulations around the world and the lack of laws in regards to cryptocurrencies, many exchanges only trade cryptocurrencies against other cryptocurrencies, e.g., you can not buy Ethereum (ETH) using U.S. dollars (USD) or your national currency, but you can buy ETH using Bitcoin (BTC). This is a way for exchanges to circumvent regulation laws and allow trading of cryptocurrencies. The few exchanges that do trade cryptocurrencies against fiat currencies are often used as entrypoints to the cryptocurrency market, an example would be buying some BTC on Coinbase (2018) using a fiat currency and then transferring those BTC to another exchange to trade cryptocurrencies.

Another important aspect in which cryptocurrency trading separates itself from most traditional markets is that the exchanges never close, the trading of cryptocurrencies can be done any time of day, any day of the year.

1.1.1 | Binance Crypto Exchange

Binance (2018a) is among the worlds largest crypto-to-crypto exchanges, both in terms of trading volume and users, with almost 8 million users at the time of writing. They offer a total of 264 trading pairs: 109 trading against BTC, 107 trading against ETH, 42 trading against their own cryptocurrency Binance Coin (BNB), and 6 trading against Tether (USDT). While trading crypto-to-crypto can be highly profitable, it can also be a risky endeavour partly due to volatility of the cryptocurrency market and partly due to the lack of regulation. Trading crypto-to-crypto is essentially trading two asset which both have highly volatile valuations in terms of fiat currency. To provide a more stable cyptocurrency, in terms of fiat value, the Tether cryptocurrency was created. The USDT creators claim to hold one USD for each UDST created, thus, the value of one USDT is tethered to one U.S. dollar, hence the name. This way USDT serves as a proxy for the U.S. dollar and allows for a more stable cryptocurrency to trade against. An example of the advanced trading interface on Binance (2018b) is shown in Figure 1.2, which contains the candlestick chart, volume chart, order book, market history, trade history, and order window.



Price(USDT)	Amount(BTC)	Total(USDT)
7660.76	0.135000	1,034.20260000
7660.47	0.052075	398.91897525
7660.39	0.019566	149.88319074
7660.03	0.582713	4,463.59906139
7660.00	1.825112	13,980.35792000
7659.52	0.188526	1,444.01866752
7659.44	0.000019	0.14552936
7657.40	0.698698	5,350.21006520
7657.39	0.104370	799.20179430
7657.00	0.545349	4,175.73729300
7656.86	0.010000	76.56860000
7655.74	0.020000	153.11480000
7655.73	0.698698	5,349.04323954
7655.00	5.089315	38,958.70632500
7,654.98 $7,654.98		
7654.99	0.001320	10.10458680
7654.98	0.092362	707.02926276
7654.97	0.000017	0.13013449
7654.00	0.412650	3,158.42310000
7652.01	0.162713	1,245.08150313
7652.00	1.008855	7,719.75846000
7651.63	0.070290	537.83307270
7651.61	0.008888	68.00750968
7651.02	1.000000	7,651.02000000
7651.00	9.356567	71,587.09411700
7650.02	0.001640	12.54603280
7650.00	0.525275	4,018.35375000
7649.81	0.007198	55.06333238
7648.52	1.564348	11,964.94696496

Price(USDT)	BTC	Time
7,654.98	0.065201	18:02:55
7,654.98	0.039051	18:02:54
7,654.97	1.739381	18:02:54
7,654.97	0.000017	18:02:54
7,654.97	0.882097	18:02:53
7,654.97	0.019000	18:02:53
7,654.97	0.040199	18:02:53
7,654.97	0.015287	18:02:53
7,654.97	0.043400	18:02:53
7,654.00	0.233807	18:02:53
7,654.98	0.001320	18:02:53
7,654.98	0.006597	18:02:53
7,654.98	0.001320	18:02:51
7,654.27	0.377105	18:02:51
7,654.27	0.336819	18:02:51
7,654.27	0.145149	18:02:51
7,654.27	0.019000	18:02:51
7,654.98	0.148326	18:02:49
7,654.99	0.207382	18:02:49
7,654.99	0.584400	18:02:49
7,654.98	0.249846	18:02:49
7,654.98	0.148326	18:02:49
7,654.98	0.036429	18:02:49
7,654.98	0.019000	18:02:48
7,654.98	0.005768	18:02:46
7,654.98	0.019000	18:02:45
7,654.98	0.100908	18:02:45
7,654.62	0.695812	18:02:45
7,654.27	0.121918	18:02:45
7,654.98	0.010442	18:02:45
7,654.98	0.031327	18:02:45
7,652.01	0.021276	18:02:44

Time	Pair	Type	Price	Filled	Total	Fee

Buy BTC -- USDT
Price: 7651.63 USDT
Amount: BTC
25% 50% 75% 100%
Total: 0.00000000 USDT
Buy

Sell BTC -- BTC
Price: 7651.63 USDT
Amount: BTC
25% 50% 75% 100%
Total: 0.00000000 USDT
Sell

Figure 1.2: The advanced trading interface on Binance (2018b), consisting of the candlestick and volume chart, order book, market and trade history, and order window.

1.1.2 | Trading on Binance

On the top left window in Figure 1.2, we see what is commonly referred to as a *candlestick* or *open-high-low-close* (OHLC) chart. Candlesticks (hereafter referred to as candles) are a representation used for aggregating price information into discrete time intervals. A candle consists of four measurements for an asset during a period: the opening price at the start of the period, the highest and lowest price within the period, and the closing price at the end of the period. The opening and closing part of a candle is usually charted as a box and the highest and lowerst prices as the "wicks" above and below. If the closing price of a candle is lower than the opening price the candle is usually coloured red, and if the opening price is lower than the closing price the candle is coloured green, as shown in Figure 1.3. Candles themselves trivially aggregate into larger candles, a 1 hour candle is for example easily created by aggregating 60 candles of 1 minute. For the remainder of the thesis we refer to candles prefaced with the abbreviated aggregation interval, thus, 1 minute and 1 hour candles become 1m and 1h candles, respectively. Throughout this thesis we make use of three order types: the

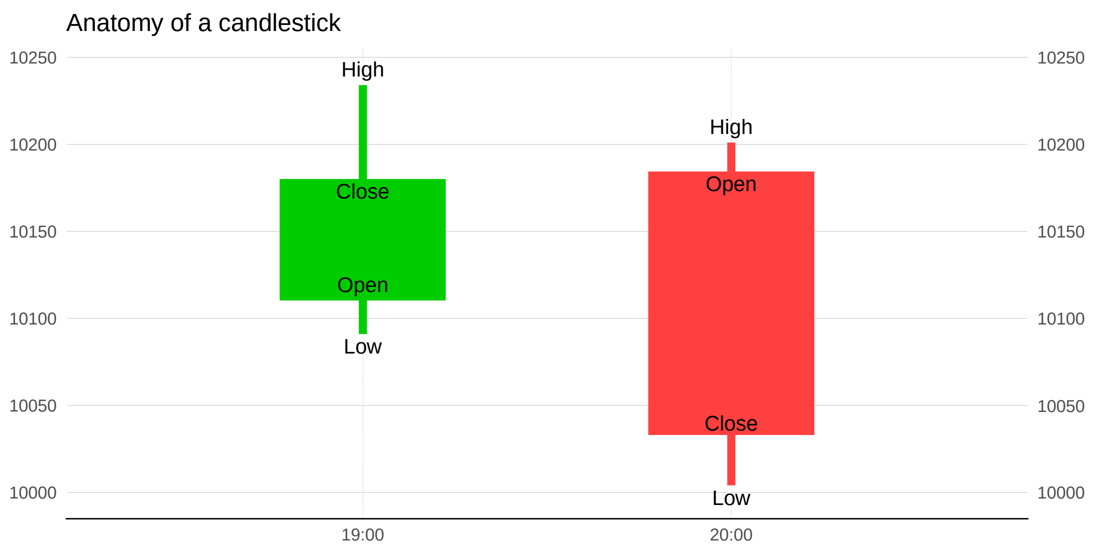
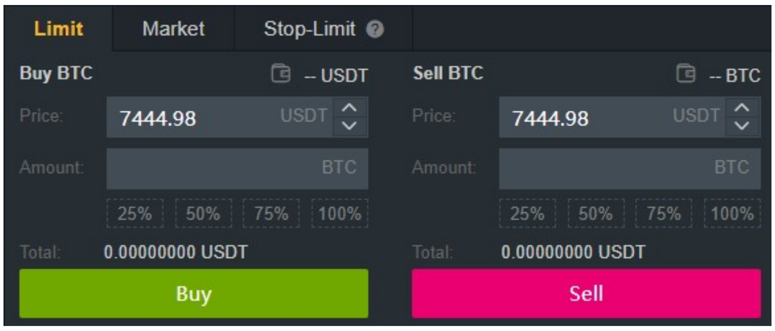


Figure 1.3: The anatomy of a candle containing the opening, highest, lowest, and closing price for the period it covers. The color of the candle shows if the closing price of the candle is above (green) or below (red) the opening price.

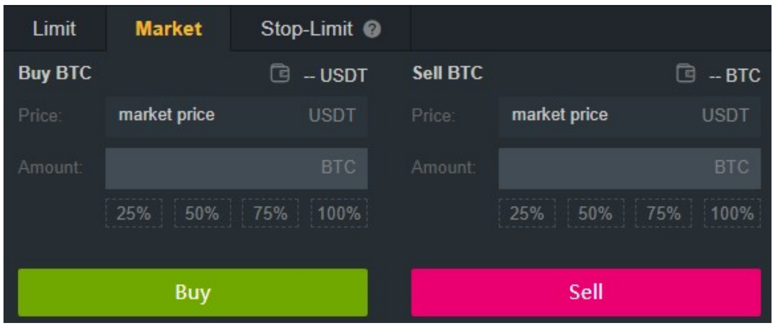
limit, market, and stop-limit order. The Binance interface for placing these orders is shown in Figures 1.4a-1.4c and described below.

- **The limit order** places an order on the order books such that when the market price reaches the specified limit, the order, or part of it, is triggered. The limit order can be used in both directions, to sell when a certain price increase has occured, or to buy when a certain price decrease has occured.

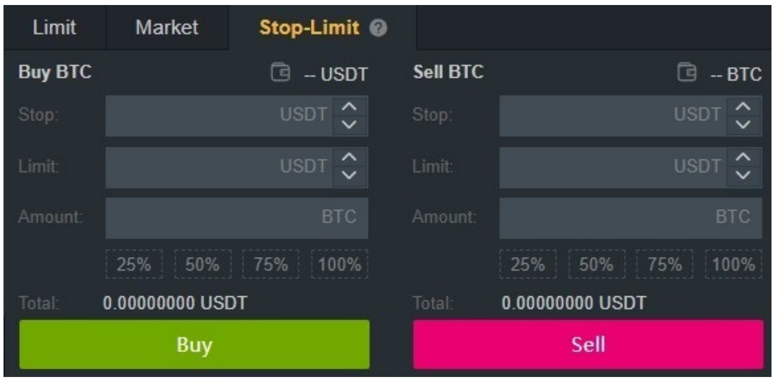
- **The market order** fulfills the orders closest to the market price on the order books until the full amount is traded, or the trading account runs out of funds.



(a) The Binance limit order interface.



(b) The Binance market order interface.



(c) The Binance stop-limit order interface.

When using market orders, caution should be applied when trading large amounts in illiquid markets, as this order type will fulfill already placed orders on the order books, meaning the price you end up paying can increase substantially from what you thought it would be.

- **The stop-limit order** is a combination of the market and limit orders in that it uses a stop price and when reached triggers a market order to either buy or sell the specified amount. A limit can then be supplied to ensure you don't buy above or sell below this.

We make three assumptions in regards to the trading simulations in this thesis.

1. When we place a market order, the full amount of the order is traded at the same price.
2. When a limit order is triggered, the full amount of the order is traded at the limit price.
3. When a stop-limit order is triggered, the full amount of the order is traded at the limit price, which is set to the same as the stop price.

1.2 | Deciding When to Trade

At the core of automated trading is some automated decision-making procedure, the derivation of which, is as we mention, the main focus of this thesis. Decision making in trading can be performed in many ways, we choose an approach where a target profit percentage is defined and subsequently attempt to predict whether this profit will be realized in the near future. This setup reduces the decision-making to a binomial classification problem, i.e., at any given time should we buy the asset under consideration or stay at our current position? Consider the price of an asset at time t, p_t, and a desired percent profit for each trade, P, we then buy at time t if the expected price at time $t+h$, p_{t+h}, is greater than or equal to $(1+P)p_t$ and stay if it is not, where h is some positive number of time steps in the future. Formally the decision can be defined as

$$Action = \begin{cases} Buy, & \text{if } E[p_{t+h}] \geq (1+P)p_t, \\ Stay, & \text{if } E[p_{t+h}] < (1+P)p_t, \end{cases} \tag{1.1}$$

thereby reducing the decision-making to the prediction of $E[p_{t+h}] \geq (1+P)p_t$, which either evaluates to true of false. To facilitate the prediction of $E[p_{t+h}] \geq (1+P)p_t$ we assume the existence of some local market dynamics that can be estimated, from which we can predict the condition. We formalize this assumption in Hypothesis 1 below.

Hypothesis 1: Local Market Dynamics

Given a set of explanatory variables X_t, we assume the existence of some time dependent function f_t, such that

$$f_t(X_t; P, h, p_t) = E[p_{t+h}] \geq (1+P)p_t,$$

given a profit limit P and horizon h.

We make no assumptions concerning the functional form of f_t other than it changes over time. We do not make any assumptions regarding the exact content of X_t we only assume existence. To test Hypothesis 1 we need to define a set of explanatory variables, X_t, or a proxy hereof, we need to decide on a target profit, P, and we need to decide the time horizon, h. Throughout this thesis we attempt to estimate f_t using supervised machine learning applied to trading data from Binance and variables derived from trading data as a proxy for X_t.

In Section 1.3 we describe how to obtain trading data through the Binance API. In Sections 1.4.1 and 1.4.2 we process the raw data in order to facilitate analysis and perform feature engineering. In Section 1.4.3 we classify the processed trading data observations in accordance with the conditions in (1.1), in order to facilitate the estimation of f_t, and further discuss choices of h and P.

1.3 | The Binance API

We obtain trading data through the Binance API (2018), specifically through the klines endpoint, which takes the parameters shown in Table 1.5. While the API documentation

	Type	Mandatory	Description
symbol	STRING	YES	
interval	ENUM	YES	
limit	INT	NO	Default 500; max 500.
startTime	LONG	NO	
endTime	LONG	NO	

Figure 1.5: The parameter inputs for the Binance API klines endpoint, as found in the Binance API (2018) documentation.

states that the maximum number of candles, for each API call is 500, we find that it actually defaults to 1000. Each candle is uniquely defined by its opening time, so for each API call we request 1000 unique 1 minute candles, a period of roughly 16.67 hours. To obtain candles in a specific interval we can supply startTime and endTime parameters, which are UNIX timestamps, i.e., milliseconds that passed since 1970-01-01 00:00:00 UTC. We set up a function that repeatedly makes the API calls necessary to obtain data for any given period. The candles returned from the API consist of the following variables

```
[
  [
    "1499040000000",      // Open time
    "0.01634790",         // Open
    "0.80000000",         // High
    "0.01575800",         // Low
    "0.01577100",         // Close
    "148976.11427815",    // Volume
    "1499644799999",      // Close time
    "2434.19055334",      // Quote asset volume
    "308",                // Number of trades
    "1756.87402397",      // Taker buy base asset volume
    "28.46694368",        // Taker buy quote asset volume
    "17928899.62484339"  // Ignore
  ]
]
```

of which we use the open time, open, high, low, close, volume, and number of trades, where the open time is a UNIX timestamp.

We collectively refer to a single observation received from the Binance API as a candle, even though volume and number of trades are not included in the definition of a candle. An example of 1m candles obtained through the Binance API is shown in Figure 1.6. The R-code used for obtaining trading data through the Binance API is found in Appendix B.2.1.

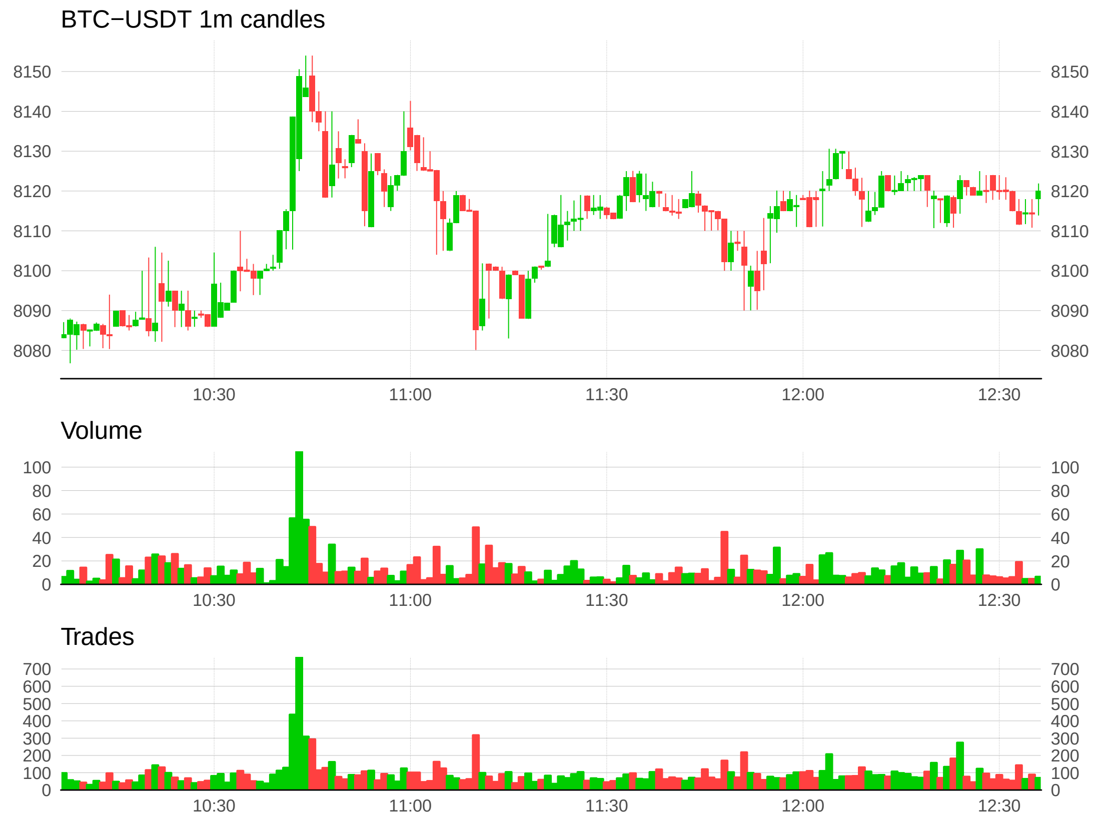


Figure 1.6: An example of the 1m candles obtained through the Binance API kline endpoint for the BTC-USDT trading pair.

1.4 | Data Preparation

The following sections describe the data preparation that we use in order to facilitate supervised learning. First we describe the candle aggregation and how we derive and add additional explanatory variables. Then how to classify the candles as either buys or stays, difference and lag the data, and split it into training, validation, and test sets. We collectively refer to these steps as *data parametrization*. We initially consider a multitude of ways in which the data parametrization can be performed subsequently we limit ourselves to number of parametrizations we can feasibly test through a greedy modelling approach. A top-level implementation of the data preparation is found in Appendix B.2.2.

1.4.1 | Aggregation

The raw 1m candles could potentially be too noisy to use for training the models. Thus, we consider aggregating the 1m candles into 11 larger intervals: 5m, 15m, 30m, 1h, 2h, 4h, 6h, 8h, 12h, and 24h.

Consider aggregating five 1m candles into a 5m candle. The opening time and price of the 5m candle is then the opening time and price of the first of the five 1m candles. The high and low price of the 5m candle is the highest and lowest price observed within any of the five 1m candles. The closing price of the 5m candle is the closing price of the last 1m candle. The volume and number of trades for the 5m candle are the sum of the trading volume and number of trades performed in the five 1m candles.

Producing aggregated candles identical to those shown on the Binance exchange for 5m, 15m, 30m, and 1h intervals is straight forward, they should simply start at times, such that the candles' opening time is at every whole 1, 5, 15, 30, and 60 minute interval. As an example, when aggregating into 30m candles, we simply start at either XX:30:00 or XX:00:00. The aggregation of the intervals larger than 1 hour, however, needs to start at specific hours in order to correctly represent the aggregation used on Binance. Further investigation of the candles on Binance show that they start these intervals at 01:00:00 CET. Aggregating the 1m candles might result in the first (last) candle containing less 1m candles than the remaining aggregated candles, as such we exclude aggregated candles if they are not "full". An example of candle aggregation is shown in Figure 1.7, which depicts BTC-USDT data over the same period aggregated into 15m, 30m, and 1h candles, respectively. The implementation of aggregating the candles is found in Appendix B.2.3.

1.4.2 | Factor Addition

After performing different levels of aggregation, we consider adding multiple factors to the data. The first factor we consider is simply the direction of each candle as a binary variable, i.e., if the price moved up during the candle the factor assumes a value of 1, and if it moved down, or stayed the same, the factor assumes a value of 0. The second factor we consider is the hour at which the candle started, as we do not include the opening time of the candles in the explanatory variables X_t, we examine the relevance of the candle opening hour through this factor.

The remaining factors we add are based on three widely-used *technical analysis* (TA) indicators: the Relative Strength Index (RSI), the Moving Average Convergence/Divergence (MACD), and the Average Directional Index (ADX). An explanation of each of these TA indicators, and how we apply them, are found in Appendix A. Since the TA factors are all based on moving averages, any dataset we add these factors to loses a number of observations at the beginning of the dataset. To ensure a fair comparison between models trained on data with and without the TA factors, we remove the same number of observations from the dataset not containing the TA factors. We base the TA factors on the *exponential moving average* (EMA) to give more weight to more recent observations than the *simple moving average* (SMA). TA factors on their own can be used for decision making in trading, we make no claims as to the effectiveness of using TA factors for trading. However, the motivation for the inclusion of TA factors is that

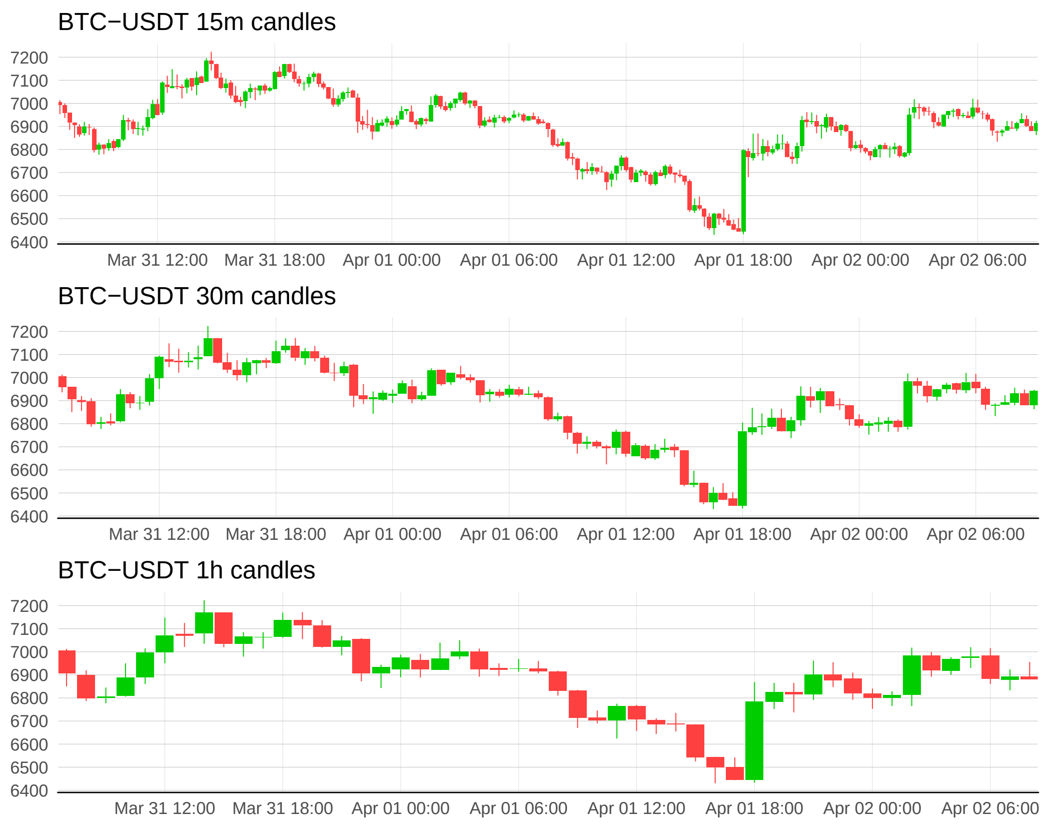


Figure 1.7: BTC-USDT 1m candles in the period from March 31st, 2018 at 07:00:00 to April 2nd, 2018 at 08:59:59 aggregated into 15m, 30m, and 1h candles.

they provide ways of summarizing historical price movements into a single signal. These five factors can be included or excluded in 32 different ways, and are summarized in Table 1.8

	Value	Total
Factors	None, Direction, Hour, RSI, MACD, ADX	32

Figure 1.8: The factors under consideration and the total number of combinations they can be included or excluded in.

1.4.3 | Classification

For this framework we need a response vector to perform supervised learning. In order to create a response vector we need a to set a desired profit limit, P, and time horizon, h, denoting within how many candles the profit should be made. Furthermore in our implementation we add a stop-limit which means that within a given horizon, h, if the the price falls below some threshold before reaching an increase of P, we should have stayed. Consider a limit of 2%, a stop-limit of 10%, and a time horizon of 24 candles, and assume we buy at the closing price of each candle. We then classify each candle in one of two ways by checking if the price at closing time t, p_t, increases atleast 2% before time $t + 24$, then check if the price decreases atleast 10% in the same period. If the price movement triggered the limit order before the it triggered the stop-limit, we consider this candle a buy. If, on the other hand, the stop-limit order is triggered before, or on the same candle as the limit order we consider this candle a stay, and likewise if none of the orders are triggered within the horizon. The candles towards the end of the dataset, for which there is not enough future candles to classify them within the time horizon, are classified as stays.

The inputs for classifying when to buy and when to stay are obviously subject to change. To find the optimal combination of limit, stop-limit, and horizon we initially consider the values for each parameter reported in Table 1.9. The R-code used for aggregating candles is found in Appendix B.2.5.

	Value	Total
Limit	0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10	10
Stop	0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10	10
Horizon	12, 24, 36	3

Figure 1.9: The values under consideration for each parameter used for classifying the candles into buys or stays and the total number of values considered for each parameter.

1.4.4 | Differencing, Lagging, and Splitting

We further consider differencing the data, i.e., we take the first difference of the open, high, low, and close prices in the datasets. The differencing is motivated by the fact that the raw values of the variables will be highly correlated whereas the first difference of the variables wont be as correlated.

We consider lagging the datasets, such that each observation in the design matrices contain all the lagged values between the current period and lagged value. We lag the open, high, low, close, volume, number of trades, and direction variables, but not the hour, RSI, MACD, and ADX factors. Consider lag 11 for example, each observation of the design matrix constructed using this lag contains the open price of the current period, as well as the open prices of the previous 11 periods, and likewise for the other lagged variables. Adding the lagged variables is motivated by the hope that some models perhaps being able to extract predictive features from the historical prices. We consider four orders of lagging the data: 0, 11, 23, and 35, and to ensure a fair comparison between models trained on raw data and lagged data we remove observations from the datasets, which use an order of lag lower than the highest we consider.

Finally we split the data into training, validation, and test sets. We set the first 60% of the observations aside as a training set, the following 20% are set aside as a validation set, and the last 20% as a test set. Since our hypothesis is that we can estimate changing local market dynamics, we also need to consider the length of the training set. For a dataset of fixed length we might want to consider reducing the percentage of data used for training to 40% or 20%, effectively dropping the first 20% or 40% of observations from the data, while maintaining validation and test sets the same size. An example of how the BTC-USDT trading pair aggregated to 1h candles is split into a 60% training, 20% validation, and 20% test set is shown in Figure 1.10. The values we consider for

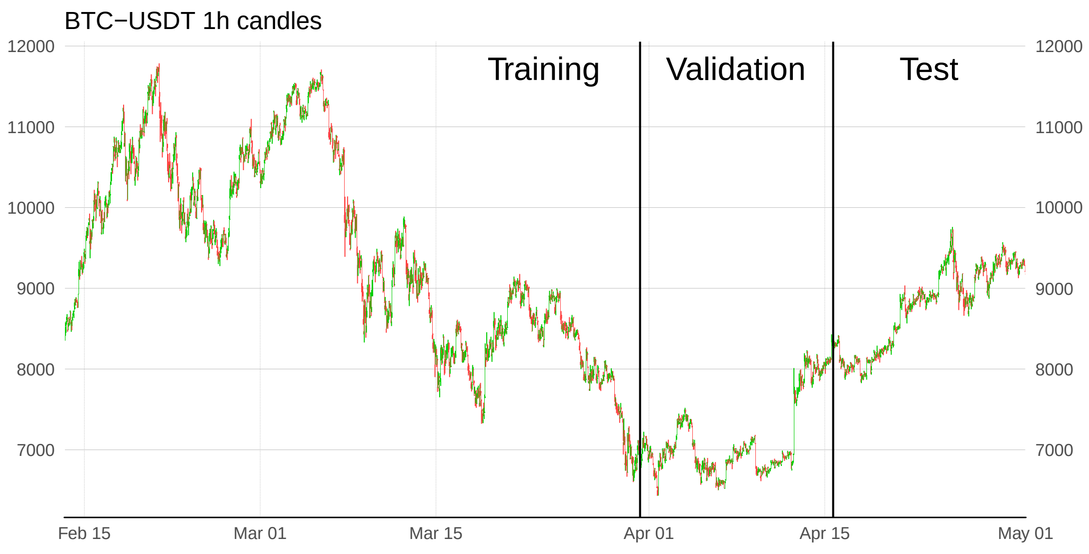


Figure 1.10: BTC-USDT aggregated to 1h candles in the period from February 11th, 2018 at 21:00 to May 1st, 2018 at 01:00 and split into a 60% training, 20% validation, and 20% test set.

each parameter of differencing, lagging, and splitting the data are summarized in Table 1.11 and the implementation hereof is found in Appendix B.2.6.

	Value	Total
Difference	0, 1	2
Lag	0, 11, 23, 35	4
Training	0.2, 0.4, 0.6	3
Validation	0.2	1
Test	0.2	1

Figure 1.11: The values for each parameter used for differencing and lagging the data, and for splitting the data into training, validation, and test sets, and the total number of values considered for each parameter.

1.5 | Limitations

Throughout the initial setup we include all values of the different parameters that we find worth considering, which results in 264 trading pairs, 11 aggregation intervals, 5 factors which can be added to data in 32 different ways, 10 limit percentages, 10 stop-limit percentages, 3 horizons, 2 difference orders, 4 lagging orders, 3 training set sizes, 1 validation set size, and 1 test set size. This leaves us with $669,081,600$ total combinations, an infeasible number of ways to conduct the analysis. In the following sections we discuss how to limit the data and parameters to consider for further analysis. The values for each parameter considered in the initial setup are reported in Table 1.12.

	Value	Total
Pair	109 BTC, 107 ETH, 42 BNB, 6 USDT	264
Interval	1m, 5m, 15m, 30m, 1h, 2h, 4h, 6h, 8h, 12h, 24h	11
Factors	None, Direction, Hour, RSI, MACD, ADX	32
Limit	0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10	10
Stop	0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10	10
Horizon	12, 24, 36	3
Difference	0, 1	2
Lag	0, 11, 23, 35	4
Training	0.2, 0.4, 0.6	3
Validation	0.2	1
Test	0.2	1

Figure 1.12: The values for each parameter used for the data parametrization of aggregating, adding factors, classifying, differencing, lagging, and splitting the data in the initial setup.

1.5.1 | Data Limitations

The first data limitation we face is obvious; not all trading pairs have been traded on Binance for the same amount of time. To get the same time period for all trading pairs we would then have to reduce the size of all datasets to that of the shortest. We restrict the number of trading pairs to only consider the pairs that are traded against USDT. These six pairs are BTC, ETH, BNB, NEO, Litecoin (LTC), and Bitcoin Cash (BCC). While some of these trading pairs have been trading on Binance since its launch, some began trading later. All of the six trading pairs have been trading since December 11th, 2017 at 01:00, so we consider this the starting date for the datasets and let them range up until May 1st, 2018 at 00:59.

Further investigating the data in the considered period reveals some time irregularities in the period between December 11th, 2017 and February 11th, 2018 creating gaps in the data. It seems only few observations are affected, which might be due to technical difficulties at Binance or problems with their API kline endpoint. An example of the time irregularities is shown in Figure 1.13, which shows the seconds between each consecutive observation for the BTC-USDT trading pair. As a result of these oberservations we cut off the datasets after the irregularities, letting them range from February 11th, 2018 at 21:00 to May 1st, 2018 at 01:00, and consist of 112501 1m observations per trading pair.

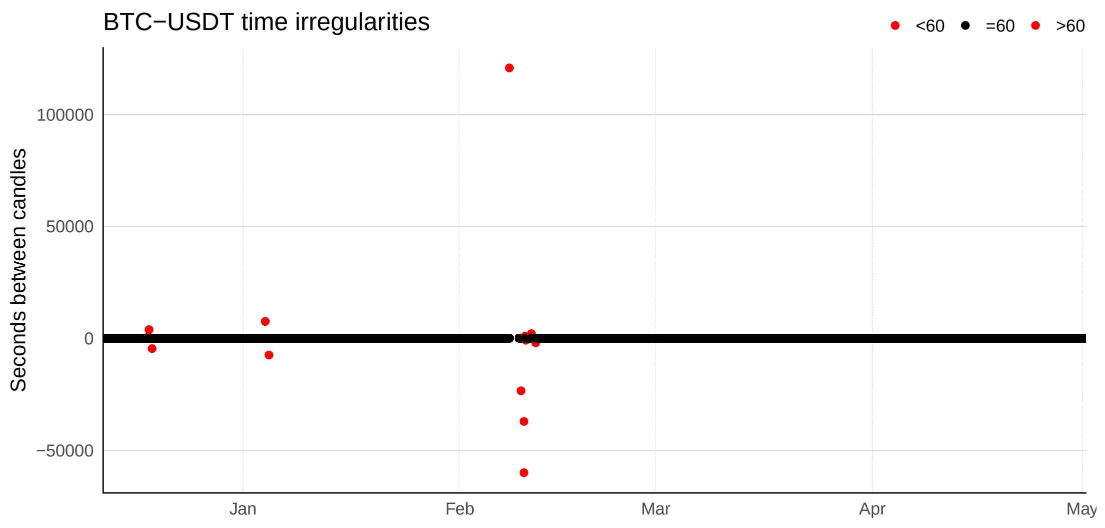


Figure 1.13: BTC-USDT time irregularities in the period December 11th, 2017 to May 1st, 2018, measured in seconds. We expect to see 60 seconds between each consecutive 1m candle.

1.5.2 | Parameter Limitations

In the bullet points below we describe the parameter limitations we perform and why.

- **Aggregation intervals** - First we exclude the 1m and 5m intervals since we fear that these might be too noisy to learn from. From here we choose only to consider 15m, 30m, and 1h candles since these aggregation levels provide a higher number of observations compared to the larger intervals while hopefully not being too noisy.

- **Factors** - In Section 1.5 we discuss how many possible ways of adding factor are available, but we decide on only two cases: include all factors or exclude all of them.

- **Limits** - While making high profits is desireable, due to the aggregation intervals we select it makes less sense to consider the higher limit values and we restrict the limits to five values: 0.01, 0.02, 0.03, 0.04, 0.05.

- **Stops** - Stop-limits too close to the buying price will be triggered if we do not hit the bottom of the price movement in a given period. We restrict the stop-limits to six values: 0.05, 0.06, 0.07, 0.08, 0.09, 0.10.

- **Horizons** - From inspecting data, paired with the profit and aggregation interval choices, 24 candles seems to be a reasonable horizon. Thus, we use a 24 candle horizon.

- **Differences** - We consider both orders of difference we set out with: 0 and 1.

- **Lags** - Higher orders of lag implies more historical information in each observation, but going too high will result in a massive, perhaps noisy, design matrix, so we restrict the orders of lag to three values: 0, 11, and 23.
- **Training set sizes** - We would like to test if the size of the training set plays a significant role in the effectiveness of the algorithms, but for now simply use the first 60% observations for training.

1.5.3 | The Restricted Setup

After restricting which trading pairs and parameter values to consider, we end up with a total of 6480 possible combinations. For each of the six trading pairs we consider three aggregation intervals, and for each of these we consider 360 different parametrizations. It is still infeasible to perform an exhaustive search across all combinations, so we perform a preliminary study of the BTC-UDST pair in Chapter 6 to further reduce the number of parametrizations. The trading pairs and parameter values considered in the restricted setup are reported in Table 1.14.

	Value	Total
Pair	BTC, ETH, BNB, NEO, LTC, BCC	6
Interval	15m, 30m, 1h	3
Factors	Included, Excluded	2
Limit	0.01, 0.02, 0.03, 0.04, 0.05	5
Stop	0.05, 0.06, 0.07, 0.08, 0.09, 0.10	6
Horizon	24	1
Difference	0, 1	2
Lag	0, 11, 23	3
Training	0.6	1
Validation	0.2	1
Test	0.2	1
Total		6480

Figure 1.14: The values for each parameter used for preparing the data by aggregating, adding factors, classifying, and differencing, lagging, and splitting in the restricted setup.

1.5.4 | Binance Fee Structure

The Binance Fee (2018) structure is set up such that for every trade performed you pay a 0.1% fee of the traded asset. Binance does provide ways of lowering the trading fees, such as paying fees throung their own cryptocurrency, Binance Coin (BNB), which lowers the fee by 50%. Further reduction can be obtained through their referral program; when trading on an account that was created through a referral link, the referrer gets 20% of all fees paid by the account. Combining these ways of reducing trading fees could theoretically reduce it to 0.4% per trade. While the simple calculations above lead to a trading fee of 0.4%, in reality, when using BNB to pay the trading fees you buy BNB at a given time at a given price and depending on the price changes in BNB itself, the trading fees are subject to change as well.

As such, we simply assume the full trading fee of 0.1%. The trading fees for a full trade then consists of the fee paid when buying the asset and the fee paid when selling the asset again, $F = 0.001 \cdot (p_t + p_{t+h})$. For simplicity we assume that the trading fees are paid on top of the amount bought per trade, i.e., if we buy \$100 worth of some asset, we actually have to pay \$100.1 for the same quantity of the asset when accounting for fees.

1.5.5 | Calculating Profits

To measure the performance of a classification algorithm on the data we set up a way to measure the profit, or loss, given the classification. A given model will classify the dataset into a number of buys and stays, for which we only need the buys to calculate the profits. For each buy we impose a time restraint for how long the trade will remain active, the time restraint used is simply the same horizon of 24 candles as used for classification. We handle the different profit and loss cases as follows.

- If the limit order is triggered before the stop-limit order within the time horizon, the profit of that trade is equal to the specified limit order percentage and the fee is $0.001 \cdot (p_t + (1 + P)p_t)$. The total profit is given by
$$p_t P - 0.001 \cdot (p_t + (1 + P)p_t).$$
- If the stop-limit order is triggered before the limit order within the horizon, the loss of that trade is equal to the specified stop-limit order percentage and the fee is $0.001 \cdot (p_t + (1 - L)p_t)$. The total loss is given by
$$-p_t L - 0.001 \cdot (p_t + (1 - L)p_t).$$
- If the limit and stop-limit orders are both triggered on the same candle, we assume the worst case scenario that the stop-limit order is triggered first and the loss of that trade is equal to the specified stop-limit order percentage and the fee is $0.001 \cdot (p_t + (1 - L)p_t)$. Which yields the same loss as in the previous case.
- If neither the limit or stop-limit orders are triggered within 24 candles, we sell at the closing price and the profit, or loss, is equal to the difference in price between the buying price and the selling price, minus the fee of $0.001 \cdot (p_t + p_{t+h})$. The total profit or loss is given by
$$p_{t+h} - p_t - 0.001 \cdot (p_t + p_{t+h}).$$

Note that the calculated profits and losses we report throughout the thesis are based on multiples of the amount of each trade, such that a profit of 1.29 is actually a 129% profit of some fixed traded amount. As such we assume that every trade is performed using a fixed amount. The R-code used for profit calculations is found in Appendix B.2.7.

2 | Generalized Linear Models

This chapter is based on parts of the following; Chapter 4 in Hastie et al. (2001), Chapter 3 in Agresti (2007). The *generalized linear model* (GLM) covers a large class of models, where the response variable, Y, is assumed to follow an exponential family distribution. A GLM can be partitioned into three components:

- *Random Component*: Identifies and assumes a probability distribution of the response variable Y. In our framework Y assumes a binomial distribution with two outcomes: buy or stay.
- *Systematic Component*: Is the explanatory variables $(x_1, x_2, \dots, x_p)$ used to construct the *linear predictor*

  $$\eta = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p.$$

  In our framework the explanatory variables are observations of candles aggregated into different intervals.
- *Link Function*: Denoted $g(\mu)$, where μ is the mean of the assumed distribution of Y, specifies the link between the random and systematic components

  $$g(\mu) = \eta.$$

  When Y assumes a binomial distribution the appropriate *logit* link function is $g(\mu) = \log(\mu/(1-\mu))$.

The GLM relies on the following assumptions:

- The data $Y_1, Y_2, \dots, Y_n$ are iid.
- The response variable, Y, assumes a distribution from an exponential family.
- For each Y a vector of covariates, $(x_1, x_2, \dots, x_p)$, exists that influences Y through a single linear predictor.
- Since maximum likelihood estimates are used for variable estimation, GLM relies on large-sample approximations.
- Goodness-of-fit measures rely on sufficiently large samples.

Our framework violates the GLM assumptions in multiple ways. We are trying to predict classified candles that are all based on correlated trading data, thus, the response variables are not independent. Furthermore, we assume that the market dynamics relating trading data to the prediction of profits changes over time, thus, the responses are not identically distributed either. Additionally GLM makes some assumptions regarding the functional form of f_t in Hypothesis 1, but since we do not make any assumptions regarding the functional form this is strictly speaking not a violation. However, we do not expect the true f_t to follow the functional form assumed by GLM, but perhaps the form assumed by GLM is a reasonable enough approximation to produce tangible results. We still include GLM in our analysis to study how it stacks up against the other machine learning algorithms that rely on fewer statistical assumptions.

2.1 | Logistic Regression

When the response variable, Y, is categorical, and we use the logit link function, we arrive at the logistic regression model discussed in this section, which is a special case of the GLM. For convenience assume that the response variable is defined as $Y \in \{0, 1\}$ with a corresponding realization $x = (x_1, x_2, \ldots, x_p)$. Let $\pi(x)$ denote the posterior probability that $Y = 1$, and let $\beta = (\beta_1, \ldots, \beta_p)$, then we can model $\pi(x)$ as

$$\pi(x) = P(Y = 1|X = x) = \frac{e^{\beta_0 + \beta^T x}}{1 + e^{\beta_0 + \beta^T x}},$$

and it follows that

$$1 - \pi(x) = P(Y = 0|X = x) = \frac{1}{1 + e^{\beta_0 + \beta^T x}}.$$

This definition of the posterior probability ensures that $0 \leq \pi(x) \leq 1$. Applying the logit link function to $\pi(x)$ yields the logistic regression model for binomial classification

$$\log\left(\frac{\pi(x)}{1 - \pi(x)}\right) = \log\frac{P(Y = 1|X = x)}{P(Y = 0|X = x)} = \beta_0 + \beta^T x. \tag{2.1}$$

From (2.1) we see that the probability, $\pi(x)$, itself is not linear in the explanatory variables but logit-transformed probability is. Furthermore, logit(π) can assume any real value even though $0 \leq \pi(x) \leq 1$. Cast in terms of the three components of a GLM we can define a logistic regression as

- *Random Component*: Y is assumed to be $\text{Binom}(1, \pi)$ and the total number of successes over N trials is $\text{Binom}(N, \pi)$.
- *Systematic Component*: The explanatory variables and corresponding parameter estimates $\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$.
- *Link Function*: The logit link funktion

$$g(\pi) = \text{logit}(\pi) = \log\left(\frac{\pi}{1 - \pi}\right).$$

2.1.1 | Fitting Logistic Regression Models

Define the set of parameters to be estimated $\theta = \{\beta_0, \beta_1, \ldots, \beta_p\}$, and denote the posterior probability by $\pi(x;\theta) = P(Y = 1|X = x;\theta)$ to emphasize the dependency on the parameter estimates. Then we can write the log likelihood over N observations as

$$\ell(\theta) = \sum_{i=1}^{N} \log(P(Y = y_i|X = x_i;\theta)),$$

which can be rewritten as

$$\ell(\theta) = \sum_{i=1}^{N} y_i \log(\pi(x_i;\theta)) + (1 - y_i)\log(1 - \pi(x_i;\theta))),$$
$$\ell(\theta) = \sum_{i=1}^{N} y_i(\beta_0 + \beta^T x_i) - \log(1 + e^{\beta_0 + \beta^T x_i}). \tag{2.2}$$

We can then maximize the log-likelihood by taking the derivative and equating it to zero

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i(y_i - \pi(x_i;\theta)) = 0. \tag{2.3}$$

Equation (2.3) can be solved using the Newton-Raphson algorithm.

2.1.2 | Regularization

To deal with the high correlation of the variables contained in the trading data, and the fact that we during the modelling procedure might sometimes include variables of questionable significance, we utilize the elastic net penalization. The penalization follows that of the `glmnet` R-package used for implementation (see Hastie and Qian (2016) and Chapter 3 in Friedman et al. (2009)). The elastic net penalty is applied by adding a combination of the L1-norm (lasso) and the L2-norm (ridge) penalties to the log-likelihood in Equation 2.3. The penalized log-likelihood is given by

$$\max_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} \left[\sum_{i=1}^{N} \left(y_i(\beta_0 + x_i^T\beta) - \log\left(1 + e^{\beta_0 + x_i^T\beta}\right)\right)\right] - \lambda\left[\frac{1}{2}(1-\alpha)||\beta||_2^2 + \alpha||\beta||_1\right],$$

where λ is the shrinkage parameter controlling the degree of penalization. The parameter α controls the combination of ridge and lasso penalization used in the elastic net. Setting $\alpha = 0$ corresponds to a pure ridge penalization and setting $\alpha = 1$ corresponds to a pure lasso regression. The ridge penalty shrinks coefficients of the correlated predictors, usually keeping all of them, at different levels of shrinkage, by using the squared values of the coefficients. The lasso penalty usually picks one of the correlated predictors by shrinking the rest to zero using the absolute value of the coefficients. The elastic net uses a combination of the two for $0 < \alpha < 1$.

3 | Neural Networks

This chapter is based on Chapter 11 in Hastie et al. (2001), Chapters 1-4 in Francois Chollet (2018), and Chapter 5 in Bishop (2006). The basic idea behind neural networks is to model some objective by filtering input variables through a sequence of linear transformations and non-linear *activation functions*. In this chapter we describe a simple neural network but the theory generalizes trivially. Figure 3.1 shows a single hidden layer neural network with the *input layer* on the left, the *hidden layer* in the middle, and the *output layer* on the right. In this chapter we first describe the neural network topography in a general K-class classification, then proceed to discuss the fitting procedure in Section 3.1. The theory described in this chapter applies to regression as well. We note that there are p input variables, H hidden layer nodes, and K outputs,

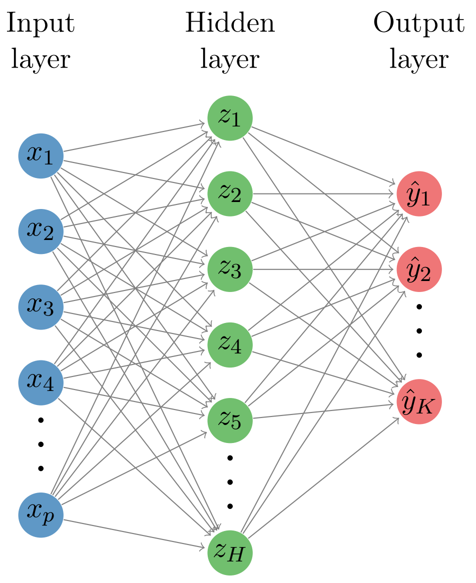


Figure 3.1: A simple K-class classification neural network with p input variables, H nodes in the hidden layer, and K probabilities returned in the output layer. The probabilities in the output layer all correspond to the probability of a given observation belonging to the respective class.

which are the probabilities of a given observation belonging to the respective class. The output variables are modelled as a functions of the derived features, z_h, in the hidden layer. A deeper neural network is obtained by adding additional hidden layers, which would be represented by additional layers of z's in Figure 3.1.

We denote the vector of predicted probabilities by $\hat{y} = f(x)$, where $x = (x_1, x_2, \dots, x_p)$ is the vector of input variables. Formally the neural network in Figure 3.1 is defined as

$$z_h = g_1(\alpha_{0h} + \alpha_h^T x), \quad h = 1, \dots, H, \tag{3.1}$$

$$f(x) = g_{2k}(\beta_{0k} + \beta_{mk}^T z), \quad k = 1, \dots, K. \tag{3.2}$$

The scalars, α_{0Hm} and β_{0k}, are known as bias terms, which are also considered weights of the network. The functions g_1 and g_{2k} are non-linear activation functions. For application we need to choose activation functions, for the hidden layer common choices are the *rectified linear unit* (ReLU), and the *sigmoid function*. The ReLU is given by

$$g_1(x) = \max(x, 0).$$

The sigmoid function is given by

$$g_2(x) = \frac{1}{(1 + e^{-x})}.$$

For the output layer, in the case of K-class classification with mutually exclusive classes, we use the *softmax* function which given some vector, $v = (v_1, v_2, \ldots, v_L)$, is defined as

$$g_{2i}(v) = \frac{e^{v_i}}{\sum_{l=1}^{L} e^{v_l}}, \quad i = 1, 2, \ldots, L.$$

The softmax function returns a vector of probabilities for each class which sums to one, the predicted class is thus the class with the highest corresponding probability. In the binary classification case we use a final layer with a single node, $K = 1$, and use the sigmod function for activation. Regression is done by running a binary classification setup without the final activation function.

3.1 | Fitting Neural Networks

Fitting a neural network is done by estimating the weights in (3.1) and (3.2) that minimize a given loss function. The total set of parameters to be estimated are given by

$$\begin{aligned} &\{\alpha_{0h}, \alpha_h; h = 1, 2, \ldots, H\}, \quad H(p+1) \text{ weights}, \\ &\{\beta_{0k}, \beta_k; k = 1, 2, \ldots, K\}, \quad K(H+1) \text{ weights}. \end{aligned}$$

For convenience we use θ to denote the total set of parameters listed above, which then comprises a total of $H(p + 1) + K(H + 1)$ weights. To estimate the model parameters we need a loss function to minimize, for K-class classification we use the deviance

$$R(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log(f(x_i)). \tag{3.3}$$

3.1.1 | Backpropagation

The generic approach used to minimize (3.3) is gradient descent, which is commonly referred to as *backpropagation* in this setting. To apply backpropagation we need to calculate the partial derivatives with respect to each of the weights involved. The following derivations show how the derivatives are calculated using arbitrary differentiable activation functions. Let us start by calculating the partial derivatives for a single observation with respect to β_k in (3.2) given by

$$\begin{aligned}
\frac{\partial R_i}{\partial \beta_{kh}} &= -\frac{\partial}{\partial \beta_{kh}} \sum_{k=1}^{K} y_{ik} \log(f(x_i)), \\
&= -\frac{\partial}{\partial \beta_{kh}} y_{ik} \log(f(x_i)), \\
&= -y_{ik} \frac{1}{f(x_i)} \frac{\partial}{\partial \beta_{kh}} g_{2k}(\beta_{0k} + \beta_k^T z_i), \\
&= -y_{ik} \frac{1}{f(x_i)} g_{2k}'(\beta_{0k} + \beta_k^T z_i) z_{ih}.
\end{aligned}$$

For the bias term, β_{0k}, we get

$$\frac{\partial R_i}{\partial \beta_{0k}} = -y_{ik} \frac{1}{f(x_i)} g_2'(\beta_{0k} + \beta_k^T z_i).$$

Now we need to calculate partials with respect to α_h in (3.1), which is more involved since they are placed earlier in the neural network. Fortunately, due to the composite form of the neural network we already did some of the work and can calculate the rest as

$$\begin{aligned}
\frac{\partial R_i}{\partial \alpha_{h\ell}} &= -\sum_{k=1}^{K} y_{ik} \frac{1}{f(x_i)} g_{2k}'(\beta_{0k} + \beta_k^T z_i) \frac{\partial}{\partial \alpha_{h\ell}} \beta_k^T z_i, \\
&= -\sum_{k=1}^{K} y_{ik} \frac{1}{f(x_i)} g_{2k}'(\beta_{0k} + \beta_k^T z_i) \beta_k^T \frac{\partial}{\partial \alpha_{h\ell}} g_1(\alpha_{0h} + \alpha_h^T x_i), \\
&= -\sum_{k=1}^{K} y_{ik} \frac{1}{f(x_i)} g_{2k}'(\beta_{0k} + \beta_k^T z_i) \beta_k^T g_1'(\alpha_{0h} + \alpha_h^T x_i) \frac{\partial}{\partial \alpha_{h\ell}} \alpha_h^T x_i, \\
&= -\sum_{k=1}^{K} y_{ik} \frac{1}{f(x_i)} g_{2k}'(\beta_{0k} + \beta_k^T z_i) \beta_k^T g_1'(\alpha_{0h} + \alpha_h^T x_i) x_{i\ell}.
\end{aligned}$$

For α_{0h} we get

$$\frac{\partial R_i}{\partial \alpha_{0h}} = -\sum_{k=1}^{K} y_{ik} \frac{1}{f(x_i)} g_{2k}'(\beta_{0k} + \beta_k^T z_i) \beta_k^T g_1'(\alpha_{0h} + \alpha_h^T x_i).$$

Assume that r iterations of gradient descent is already performed, then we update the current variable estimates using the partial derivatives. The gradient descent update at the $(r+1)$'th iteration is given by

$$\begin{aligned}
\beta_{kh}^{r+1} &= \beta_{kh}^{r} - \eta_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{kh}^{r}}, \\
\beta_{0k}^{r+1} &= \beta_{0k}^{r} - \eta_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{0k}^{r}}, \\
\alpha_{h\ell}^{r+1} &= \alpha_{h\ell}^{r} - \eta_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{h\ell}^{r}}, \\
\alpha_{0h}^{r+1} &= \alpha_{0h}^{r} - \eta_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{0h}^{r}},
\end{aligned}$$

where η_r is the learning rate. Backpropagation works in two steps: the first step is a forward sweep that keeps the weights fixed, while propagating the training observations through the network to produce predictions and calculate prediction errors. In the second step the prediction errors are then propagated back through network and used to update the parameter estimates. When applying backpropagation the data is usually split into batches, each batch is then passed through the backpropagation algorithm, a full data pass is then reached once all the batches have been passed through the algorithm. Usually, multiple passes of the data are used to properly estimate the parameters, the number of passes are referred to as epochs. The more epochs used the closer the training data is fitted, however, to ensure proper generalization one ideally monitors training and validation errors to decide the optimal number of epochs.

Considering that there are $H(p+1)+K(H+1)$ weights to estimate, which can quickly become a large number, gradient descent might at first seem infeasible due to the amount of partial derivatives to be calculated. But as seen above the compositional model form actually simplifies the calculation of the required gradients, allowing for gradient descent to be applied to minimize the cross entropy. The ReLU is not differentiable in zero, but the derivatives can still be calculated using sub-derivatives, which also makes for cheaper gradient calculations compared to the sigmoid function. Another desirable property of the ReLU activation function is the ability to zero out nodes, promoting sparsity in the neural network.

The weights cannot start at zero and are initialized using small random values, the backpropagation algorithm does not converge if the weights started at zero. Initializing the weights at small values greatly increases the demand for standardized input during implementation, as such, we scale data to have zero mean and unit variance when needed.

3.1.2 | Regularization

As we mention, a neural network can have many parameters, so if one obtains a global minimization of $R(\theta)$ overfitting is an imminent danger. The easiest way to avoid overfitting is to keep the amount of layers and nodes small, the amount of layers and nodes in a neural network is often referred to as *capacity*. A model with a too high capacity might learn training-specific patterns which may lead to bad generalization. Conversely, a model with too low a capacity might not capture all relevant signals in the data, and might perform poorly in both training and generalization. We can control overfitting by keeping the neural network simple and monitor the number of epochs, as we mention in Section 3.1.1.

Additional regularization can be obtained through weight regularization, specifically we can add a penalty term, $J(\theta)$, to $R(\theta)$, such that

$$R^*(\theta, \lambda) = R(\theta) + \lambda J(\theta),$$

where λ is a tuning parameter which can be estimated by cross validation, this method is referred to as *weight decay.* The penalty term is added layer-wise during implementation. For $J(\theta)$ we have some options, namely the L1-norm regularization (lasso) and the L2-norm regularization (ridge), or a combination of the two (elastic net).

The L1-norm regularization in the simple neural network is given as

$$J_{L1}(\theta) = \sum_{h\ell} |\alpha_{h\ell}| + \sum_{kh} |\beta_{kh}|,$$

which is just the sum of absolute values of all of the weights, except the bias terms. The L2-norm regularization is given by

$$J_{L2}(\theta) = \sum_{h\ell} \sqrt{\alpha_{h\ell}^2} + \sum_{kh} \sqrt{\beta_{kh}^2}.$$

Finally we can combine the two to obtain the elastic net penalization

$$J_{L1,L2}(\theta) = (1-\alpha) J_{L2}(\theta) + \alpha J_{L1}(\theta),$$

where α is a tuning parameter that controls the balance between ridge and lasso penalization.

Another popular and highly effective regularization scheme for neural networks is adding *dropout.* Adding a dropout is done by zeroing out output features of a given layer during training. At test time, the output features wont be zeroed out, thus, the output of the layer is then scaled down by the dropout rate to accommodate the fact that more nodes are activate. The intuition behind this scheme is inspired by the way tellers in some banks are repeatedly moved around, thus, requiring cooperation between tellers to successfully defraud the bank. In the neural network, randomly zeroing outputs in a layer helps prevent the model from picking up on insignificant signals.

4 | Tree Based Algorithms

In this chapter we describe the tree based machine learning algorithms used in this thesis. Trees and how to grow them are described in Section 4.1, after which we proceed to introduce the concept of boosting in Section 4.2. Finally we describe the tree based algorithms used for application; the gradient boosting algorithm is introduced in Section 4.3 and the random forest algorithm in Section 4.4. Throughout this chapter assume data is given by $(x_i, y_i),\ i = 1, 2, \ldots, N$, where N is the number of observations, $y_i \in \{1, 2, \ldots, K\}$ is the class of the i'th observation, and $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ is a vector of p explanatory variables.

4.1 | Classification Trees

In this section we discuss how to construct a classification tree, which a subclass of what is commonly known as *classification and regression trees* (CART). Classification trees work by partitioning the feature space into a set of regions and assigning classes to each region. To grow a tree we need a way to automatically partition the feature space and assign constants to the resulting regions. Figure 4.1 depicts how the feature space can be partitioned into four regions using three continuous variables and corresponding split variables. Once the regions, also referred to as terminal nodes, have been defined a

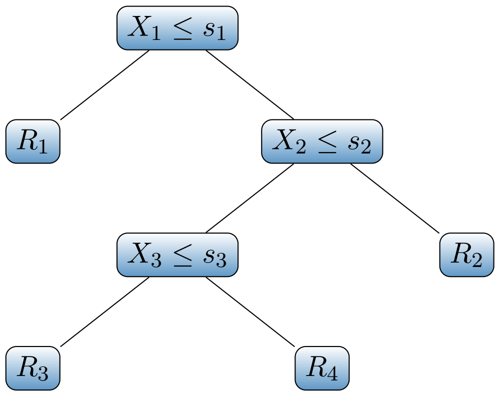


Figure 4.1: A simple tree showing how a feature space is split into four regions using three continuous variables and corresponding split points.

class is assigned to each region according to the majority class of the particular region. The splits before the terminal regions are also referred to as nodes. Trees are generally constructed by performing the following two steps.

1: Grow a large tree, which we denote T_0, stopping only when the number of observations in the terminal nodes are below a certain threshold.

2: To prevent overfitting the tree is pruned, which is generally done by collapsing nodes if their contribution to the overall model training classification accuracy is minimal.

In Section 4.1.1 we cover the process of growing a tree and in Section 4.1.2 we describe the pruning process.

4.1.1 | Growing

The challenging part of growing trees is deciding on how to partion the feature space, which is done by selecting sets of variables and associated split points. To figure out which variables to split, and how to split them, a greedy approach is taken. Starting with all data, consider the splitting variable j and splitting point s, which defines the two half-planes

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j > s\}. \tag{4.1}$$

We seek pairs (j,s) such that the resulting regions R_1 and R_2 are as pure as possible in terms of classes. To formalize the concept of pure an impurity measure is needed. Assume that the feature space is partitioned into M regions $R_1, R_2, \dots, R_M$, then define

$$N_m = \#\{x_i \in R_m\},$$
$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}(y_i = k),$$

where the $\#$ operator counts the number of observations in a given region, $\mathbb{1}$ is the indicator function. We have defined $\hat{p}_{mk}$ as the proportion of class k observations in node m, a class is assigned to a given node as $k(m) = \arg\max_k \hat{p}_{mk}$. From here different impurity measures can be defined, the following are common choices.

Misclassification error:

$$\frac{1}{N_m} \sum_{i \in R_m} \mathbb{1}(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}.$$

Gini index:

$$\sum_{k \neq c} \hat{p}_{mk}\hat{p}_{mc} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$

Cross-entropy (deviance):

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}).$$

Regardless of which impurity measure is chosen, denote it by $Q_m(T)$. We can now formally define the optimal choices of split variable j and split point s as the solution to

$$\min_{j,s} \left[N_1 Q_1(T; R_1(j,s)) + N_2 Q_2(T; R_2(j,s))\right]. \tag{4.2}$$

Where N_1 and N_2 denote the number of observations in the child nodes of the split. It is usually feasible to simply scan through all inputs to determine the pair (j, s) that minimizes (4.2). The classification tree is then grown by repeatedly using (4.2) to choose pairs (j, s) to partition the feature space, until the number of observations in the terminal nodes drop below a certain threshold. We denote a fully grown tree by T_0, which has the form

$$T_0(x_i) = \sum_{i=1}^{M} k_i \mathbb{1}(x_i \in R_i),$$

where k_i is the class assigned to region R_i and M is the number regions.

4.1.2 | Pruning

Assume that we have grown a large tree, T_0, then define a subtree, $T \subset T_0$, as any tree obtained by pruning T_0. Pruning is done by collapsing any number of non-terminal nodes. To decide which non-terminal nodes to collapse cost complexity pruning can be performed. Define the cost complexity criterion as

$$C_\lambda(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \lambda |T|, \tag{4.3}$$

where $|T|$ denotes the number of terminal nodes in T. The tuning parameter, $\lambda \geq 0$, governs the trade-off between sparsity and goodness-of-fit. The idea is to compute the tree that minimizes (4.3) for each λ. For each λ it can be show that is is possible to find a unique subtree, T_λ, that minimizes (4.3), see (Hastie et al., 2001, p. 308). For a given λ we identify T_λ using weakest link pruning. Weakest link pruning constructs a sequence of trees through an iterative procedure, where the weakest contributing node is collapsed at each step. The node with the weakest contribution is the node for which a collapse would cause the smallest increase among all nodes in

$$\sum_{m=1}^{|T|} N_m Q_m(T).$$

The resulting sequence of subtrees contains a unique smallest subtree that minimizes (4.3). Cross-validation can be applied to estimate λ and we denote the final tree $T_{\hat{\lambda}}$.

4.2 | Boosting

Boosting is based on the idea that a set of classifiers can be combined into a "committee" with a better classification performance than any of the individual classifiers. To introduce the concept of boosting we start by discussing the AdaBoost.M1 algorithm in Section 4.2.1 and then proceed to describe *gradient boosting* in Section 4.3. In Section 4.1 we use M to denote the number of tree regions, however, from here we use it to denote boosting iterations.

4.2.1 | AdaBoost

Given a binary response variable $Y \in \{-1, 1\}$ and a set of explanatory variables X, a classifier $G(X)$ that predicts either -1 or 1 based on X can be constructed. The training error rate is given by

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(y_i \neq G(x_i)). \tag{4.4}$$

Say the classifier $G(X)$ is only slightly better than random guessing then we refer to it as a *weak* classifier. The boosting procedure constitutes the application of a weak learner to modified data in a sequential manner, producing a sequence of weak classifiers $G_m(x), m = 1, 2, \dots, M$, which is the committee. To obtain a final prediction, each member of the committee gets to place a weighted vote on the prediction outcome, where higher weights are assigned to more accurate predictors. Formally the final prediction has the form

$$G(x) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right).$$

The aforementioned data modification is performed by applying weights, $w_1, w_2, \dots, w_N$, to the training observations, $(x_i, y_i),\ i = 1, \dots, N$. Initially the weights are all $1/N$, and as such, in the first the step the learner is applied to data in the usual manner. For each subsequent iteration, $m = 2, 3, \dots, M$, the weights for each observations are updated and the learner is applied to the modified data. The weights are calculated such that at iteration m the weights are higher for the observations misclassified by $G_{m-1}(x)$. As such, the final weights reflect the classification difficulty presented to the sequential set of weak learners by the respective observation. The Adaboost.M1 algorithm is described in Algorithm 1, as presented in (Hastie et al., 2001, p. 339).

4.2.2 | Boosting Trees

In Section 4.1 we show how to grow a tree, in this section we show how boosting is applied to trees. Formally a tree can be defined as

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j \mathbb{1}(x \in R_j),$$

with parameters $\Theta = \{\gamma_j, R_j\}_{j=1}^{J}$. In the classification setup, γ_j is the class assigned to observations in region R_j. The boosted tree model creates a sequence of trees that are then summed

$$f_M = \sum_{m=1}^{M} T(x; \Theta_m).$$

To estimate f_M we proceed in a forward stagewise manner. At each step the algorithm must estimate the parameter set Θ_m, conditional on the previous model, by solving

$$\hat{\Theta}_m = \min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)), \tag{4.5}$$

Algorithm 1: Adaboost.M1

```
1 - Initialize observation weights as $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2 - For $m = 1, 2, \ldots, M$ :
    (a) - Apply the weights, $w_i$, to data and use the weighted data to train the
          classifier $G_m(x)$.
    (b) - Compute the error at step $m$ as
          $$err_m = \frac{\sum_{i=1}^N w_i \mathbb{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
    (c) - Compute
          $$\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right).$$
    (d) - Use $\alpha_m$ to update the data weights by setting
          $$w_i = w_i e^{\alpha_m \mathbb{1}(y_i \neq G_m(x_i))}, \quad i = 1, 2, \ldots, N.$$
3 - After $M$ iterations we arrive at the classification model
    $$G(x) = sign\left(\sum_{m=1}^M \alpha_n G_m(x)\right).$$
```

where L is some loss function. That is, at each step we have to estimate $\Theta = \{\gamma_j, R_j\}_{j=1}^J$ conditional on the current model, f_{m-1}. Given the regions, R_j, estimating the constant in each region is typically done by solving

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i), \gamma_{jm}).$$

Using the deviance as the loss function in (4.5) turns the minimization into a difficult optimization problem, to solve (4.5) we need a fast approximative solution.

4.3 | Gradient Boosting

Solving (4.5) simplifies in some cases, see (Hastie et al., 2001, p. 357), however, if one wishes to use a loss function such as the deviance, numerical optimization is needed. In this section we cover the gradient boosting method for solving (4.5) using the deviance loss function. First we cast the problem as a numerical optimization problem solvable by steepest descent, then we argue that steepest descent might lead to poor generalization, and proceed to describe the gradient boosting method.

4.3.1 | Numerical Optimization

In this section we show how to solve (4.5) using any differentiable loss function. Consider the loss function as a function of the induced trees

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)). \tag{4.6}$$

The goal is to minimize (4.6), which if we ignore the fact that f is restricted to be trees, can be considered a numerical optimization problem

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f}). \tag{4.7}$$

The "vector of parameters", $\mathbf{f} \in \mathbb{R}^N$, consists of the values of the function at each data point

$$\mathbf{f} = \{f(x_1), f(x_2), \dots, f(x_N)\}. \tag{4.8}$$

Numerical optimization procedures solve (4.7) as a sum of component vectors

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m, \quad \mathbf{h}_m \in \mathbf{R}^N, \tag{4.9}$$

where $\mathbf{f}_0 = \mathbf{h}_0$ is an initial guess, and each successive $\mathbf{f}_m$ is induced based on the previous model, $\mathbf{f}_{m-1}$. The chosen numerical optimization method for solving (4.7) dictates how the components $\mathbf{h}_m$ are chosen.

Steepest descent can be used for minimizing (4.7), which implies $\mathbf{h}_m = -\rho_m \mathbf{g}_m$ where ρ_m, also referred to as the step length, is a scalar and $\mathbf{g}_m \in \mathbb{R}^N$ is the gradient of $L(\mathbf{f})$. The components of the gradient are given by

$$g_m = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}, \tag{4.10}$$

and ρ_m is

$$\rho_m = \arg\min_{\rho} L(\mathbf{f_{m-1}} - \rho \mathbf{g_m}). \tag{4.11}$$

After calculating the step direction, (4.10), and the step length, (4.11), the current model is updated

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m.$$

Steepest descent can be considered a greedy approach since the negative is the local direction in which the loss function decreases the most. If the ultimate goal is to minimize training error then steepest descent would be a great strategy, however, since the gradient is only defined at the data points in the training set we may end up with poor generalization.

4.3.2 | Gradient Boosting for Classification

Assume that y_{ik} is encoded such that it assumes a value of 1 if the i'th observation belongs to class k and 0 otherwise. Gradient boosting for a K-class classification tree works by inducing K coupled regression trees, $T_{km}(x; \Theta_m)$, at each iteration with predictions denoted $\mathbf{t}_{km}$, which are as close as possible to the negative gradient. Close-as-possible is measured using the squared error, which implies the minimization

$$\tilde{\Theta}_{km} = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{ikm} - T(x_i; \Theta))^2. \tag{4.12}$$

Each of the K coupled trees are fitted to its respective negative gradient given by

$$\begin{aligned} -g_{ikm} &= \frac{\partial L(y_i, f_{1m}(x_i), \ldots, f_{1m}(x_i))}{\partial f_{km}(x_i)}, \\ &= y_{ik} - p_k(x_i), \end{aligned}$$

where the probability $p_k(x_i)$ is given by

$$p_k(x_i) = \frac{e^{f_k(x_i)}}{\sum_{l=1}^{K} e^{f_l(x_i)}}. \tag{4.13}$$

Even though each of the induced regression trees are fitted separately, they are all coupled through (4.13), i.e., $\tilde{\Theta}$ can be obtained by fitting a regression tree to the negative gradient values. Algorithms for quick regression tree induction already exist, see (Hastie et al., 2001, p. 359), so we can easily solve (4.12). Solving (4.12) provides the regions of the induced tree, $\{\tilde{R}_{jm}\}_{j=1}^{J_m}$, which is the hard part. The constant in those regions are estimated to minimize (4.12), which is not the final goal so the constants are recalculated. The recalculated constant should minimize the total deviance across classes and observations, this minimization does not have a closed form solution and we settle for an approximation performed using a single Newton-Raphson step, for details see (Friedman, 2001, p. 11). The approximative solution for updating the region constant is given by

$$\hat{\gamma}_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in \tilde{R}_{jkm}} r_{ikm}}{\sum_{jkm} |r_{ikm}|(1 - |r_{ikm}|)}, \quad j = 1, 2, \ldots, J_m.$$

The regions, $\{\tilde{R}_{jm}\}_{j=1}^{J_m}$, that solve (4.12) will not be identical to the regions, $\{R_{jm}\}_{j=1}^{J}$, that solve (4.5) but generally the regions will be similar enough to serve the same purpose.

Algorithm 2 describes the gradient boosting procedure for K-class classification, as presented in (Hastie et al., 2001, p. 387). We start out by initializing a model with all probabilities equal to zero before entering the boosting iterations. The boosting procedure is performed M times in step 2 and comprises two steps. First step of each boosting iteration is defining the probability calculation, done in step (a), and in step (b) we then grow the K regression trees coupled by the probability calculations defined in step (a). Each of the K regression trees are grown in steps i-iii, first the target r_{ikm}

is computed, this is the gradient using deviance loss, see (Hastie et al., 2001, p. 360). Once r_{ikm} is computed the terminal regions, R_{jkm}, are found by fitting a regression tree to r_{ikm}. For each terminal node the associated constant is calculated and finally the model is updated.

Algorithm 2: Gradient Boosting for K-Class Classification

```
1 - Initialize $f_{k0}(x) = 0,\ k = 1, 2, \ldots, K$.
2 - For $m = \{1, 2, \ldots, M\}$:
    (a) - Set
          $$\left[ p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}} \right]_{f_k = f_{k,m-1}}, \quad k = 1, 2, \ldots, K.$$
    (b) - For $k = 1$ to $K$:
        i - Compute $r_{ikm} = y_{ik} - p_k(x_i),\ i = 1, 2, \ldots, N$.
        ii - Obtain the terminal regions, $R_{jkm},\ j = 1, 2, \ldots, J_m$, by fitting a
             regression tree to the targets, $r_{ikm},\ i = 1, 2, \ldots, N$.
        iii - Compute the terminal node values
              $$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} r_{ikm}}{\sum_{jkm} |r_{ikm}|(1 - |r_{ikm}|)}, \quad j = 1, 2, \ldots, J_m.$$
        iv - Update $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} \mathbb{1}(x \in R_{jkm})$.
4 - Output $\hat{f}_k(x) = f_k M(x),\ k = 1, 2, \ldots, K$.
```

4.3.3 | Regularization

For practical application of Algorithm 2 we still need to decide the number of boosting iterations and the number of terminal nodes for each tree, J_m. Typically a constant number, $J = J_m$, of terminal nodes for each tree grown during the boosting procedure is chosen. The number J controls the number of variable interactions. It is generally only worth considering the range $2 \le J \le 10$, see (Hastie et al., 2001, p. 363). The number of boosting iterations, M, controls how well the model fits the training data. However, as the training error is reduced, the generalization of the model eventually deteriorates as well. Thus, there exists some M^* that balances goodness-of-fit and generalization. To estimate M^* one typically inspects the error on a validation set as the number of boosting iterations is increased.

We can further impose regularization by scaling the contribution of each induced tree by a factor of $0 \le \upsilon \le 1$. The scaling is imposed in step iv of Algorithm 2 where scaled updates are performed as

$$f_{km}(x) = f_{k,m-1}(x) + \upsilon \sum_{j=1}^{J_m} \gamma_{jkm} \mathbb{1}(x \in R_{jkm}).$$

The scalar υ is commonly referred to as the learning rate in this setting. Thus, we can regulate the model using both M and υ, however, the two do not operate independently. Smaller υ typically requires a larger number of boosting iterations. In (Hastie et al., 2001, p. 363) they state that empirically the preferred strategy appears to be obtained by setting a small υ and then select M by inspecting the performance on a validation set.

4.4 | Random Forests

Random forests is a modified *bagging* procedure, which works by reducing model variance by averaging a set of de-correlated trees. In this section we first provide a brief recap of the bagging procedure and proceed to describe how this procedure is extended to the random forest.

The bagging procedure constitutes averaging the output of a set of models fitted to bootstrapped samples of the data. Assume the training data is given by $\mathbf{Z} = \{(x_1, y_2), (x_2, y_2), \ldots, (x_n, y_n)\}$, denote by $\mathbf{Z}^{*b}$, $b = 1, 2, \ldots, B$, a set of bootstrapped samples from $\mathbf{Z}$. Denote by $\hat{f}^{*b}(x)$ a model fitted to the b'th bootstrap sample, then the bagging estimate is given by

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x).$$

The motivation is that the variance of some approximately unbiased model can be reduced by creating a single model that averages a large set of models, fitted to different boosting samples.

Trees, which if grown sufficiently deep, have relatively low bias while simultaneously having a large variance, greatly benefit from averaging. Since the trees generated through bagging are identically distributed the expectation of an average of trees is the same as the expectation of a single tree, thus, improvement is obtained through a reduction of variance. An average of B identically distributed variables with pairwise positive correlation, ρ, have a variance of

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \tag{4.14}$$

As the number of bootstrapped samples, B, increases the last term disappears, leaving only $\rho\sigma^2$. The benefit of bagging is then limited by the variance and the model correlation. We mention that random forests grow de-correlated trees which reduce ρ, thereby increasing the potential benefits of bagging. When growing the trees random forests reduce the inter-tree correlation by selecting $m \leq p$ input variables before each split, where p is the total number of input variables. In Algorithm 3 we describe the

Algorithm 3: Random Forest for Classification

```
1 - For $b = \{1, 2, \ldots, B\}$:
    (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
    (b) Grow a random-forest tree $T_b$ to the bootstrap sample by recursively
        applying the following steps to each terminal node until some minimum
        node-size, $n_{min}$, is reached.
          i - Select $m$ variables among the $p$ input variables.
         ii - Pick the best variable and split-point combination from the $m$
              variables.
        iii - Split the node into two child nodes.
2 - Output the ensemble of trees $\{T_b\}_1^B$.
3 - Let $\hat{C}_b(x)$ be the predicted class by the $b$th random forest tree. The random
    forest prediction is then $\hat{C}_{rf}^B(x) = \text{majority vote}\{\hat{C}_b(x)\}_1^B$.
```

random forest algorithm, as presented in (Hastie et al., 2001, p. 588). The first step consists of two substeps, first data for growing is bootstrap-sampled in step (a) and then trees are fitted to the bootstrap sample in step (b). The second step outputs the ensemble of trees grown, and from the ensemble a majority vote decides the random forest predictions in the third step.

The restriction on the number of input variables used at each split can introduce some bias in the random forest trees. The amount of bias depends on the true underlying function, but generally as m decreases, the bias of the individual trees increases. Any improvement obtained by random forests over traditional trees are therefore solely obtained through variance reduction. A typical choice of m for classification is $m = \sqrt{p}$.

4.4.1 | Out of Bag Error

An important feature of the random forest algorithm is the ability to perform the *out of bag* (OOB) error estimates. The OOB error estimate can be obtained for each observation in the training data in the following manner:

1 - For the i'th training observation, (x_i, y_i), select all random forest trees from the ensemble $\{T_b\}_1^B$, that never saw (x_i, y_i) during training.

2 - Use the subset of trees that never saw (x_i, y_i) during training to perform a prediction and calculate the error.

The above two steps estimate the prediction error on "unseen" data, which works as a great proxy for the test error.

5 | Model Fitting

In this chapter we cover the steps used to implement a neural network (NN), gradient boosting (GB), and random forests (RF). We further comment on the application to trading data for each model. In order to illustrate the modelling, we use the IMDb dataset included in the `Keras` R-package. The dataset consists of a training and test set each containing 25000. For NN and GB we set aside 10000 observations from the training set for validation and initially train the model on the remaining 15000 training set observations. The goal is to predict whether an IMDb movie review is positive or negative. The explanatory variables are binary indicators of whether a specific word is present in the review. To limit the number of binary indicators we only consider the 10000 most popular words. Since all the variables are binary indicators no scaling is needed. The NN fitting, which is the most involved of three, is presented first, followed by GB and RF. The code required to reproduce the IMDb examples is found in Appendix B.3. At the end of the chapter we discuss potential benefits of changing the classification threshold for trading application.

5.1 | Fitting a Neural Network

The implementation is performed using the Keras framework in the `Keras` R-package. Keras is a high-level library with the necessary building blocks for fitting any kind of NN, Keras further allows for seamless integration with different back-ends for differentiation and tensor manipulation. For this project we use the Tensorflow back-end developed by Google. In Chapter 3 we cover the theory of neural networks and in this section we show how to apply them.

5.1.1 | Model Topography

First we need to define the layers of the model and since we only work with densely connected layers, we only need to specify the number of layers and width of each. If we were working with convolutional neural networks the topography can be much more complex, for further reading see Chapter 5 in Francois Chollet (2018). The following lines of code define a neural network named `model`, which consists of three layers: two hidden layers and an output layer. The two hidden layers consist of 16 nodes each, and use the ReLU activation function discussed in Chapter 3. Since the dataset is stored in a matrix, which is also referred to as a 2D tensor, we further define the shape of the input first hidden layer and only supply the number of columns as shape. The output layer, a classification layer in this case, only has a single node and uses the sigmoid activation function.

```
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
              input_shape = ncol(trainx_scaled)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

This is also the step where we can add different types of regularization. The following lines show how to add dropout regularization, with a dropout rate of 50%, to the weights in the first hidden layer.

```r
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
              input_shape = ncol(trainx_scaled)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

The other regularization options are obtained by adding either `regularizer_l1(l = 0.01)`, `regularizer_l2(l = 0.01)`, or `regularizer_l1_l2(l1 = 0.01, l2 = 0.01)` in place of `layer_dropout`.

5.1.2 | Compile Configuration

Currently the model consists of nothing but a definition of layers, which is not quite enough to build a complete model. We now define the desired model learning process, which is sometimes referred to as the compilation step. First we define the optimizer, we use rmsprop, which is a backpropagration implementation that scales the learning rate by a running average of the gradients calculated in previous iterations. The loss function used is the binary cross-entropy (deviance). Finally we define the metrics to be measured during training, in addition to measuring the loss. Note the slightly unorthodox syntax, when it comes to R, in which the model previously defined is configured inplace.

```r
model %>%
  compile(optimizer = "rmsprop",
          loss = "binary_crossentropy",
          metrics = c("accuracy")
  )
```

5.1.3 | Fitting

The model can now be trained on the training set and the validation set is then used to monitor loss and accuracy on unseen data during fitting. The training data is supplied in batches of size 512 and we run 20 epochs.

```r
history <- model %>%
  fit(trainx,
      trainy,
      epochs = 20,
      batch_size = 512,
      validation_data = list(valx_scaled, valy)
  )
```

The training results are stored in the object called `history`. In Figure 5.1 we see the loss and accuracy in both the training and validation sets plotted against the number of epochs. From Figure 5.1 we see that the training loss is steadily decreasing and accuracy increasing, however, the validation accuracy and loss indicates that after 4-5 epochs we start overfitting.

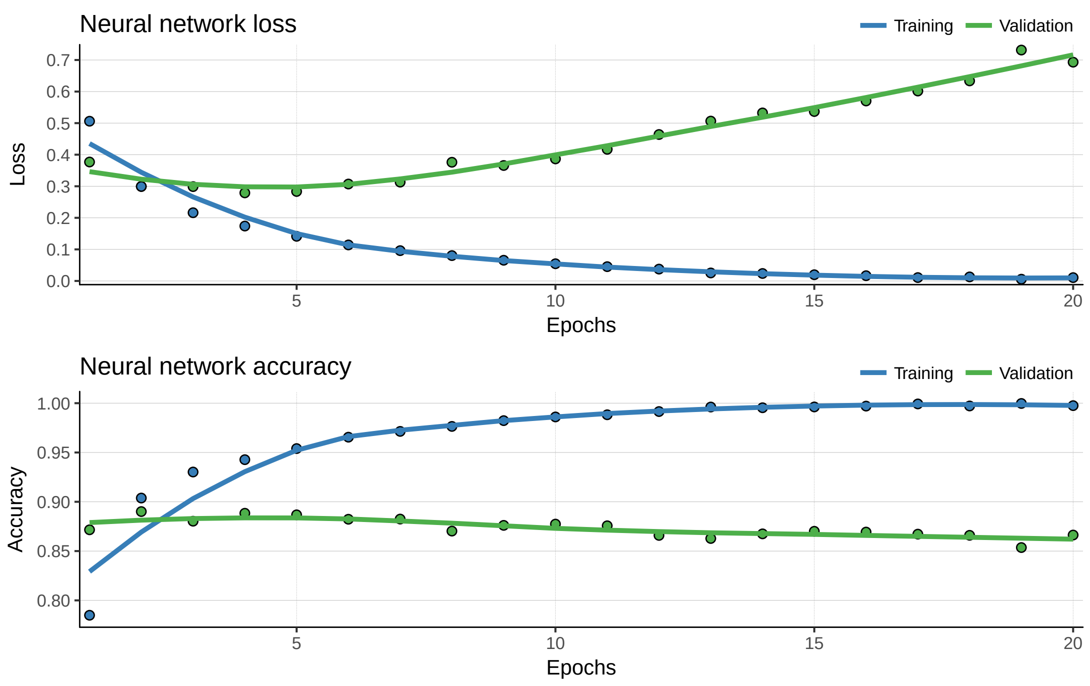


Figure 5.1: The training and validation loss and accuracy, plotted against the number of epochs, in the IMDb review classification example using neural networks.

5.1.4 | Testing

Since it seems the model starts overfitting at around 5 epochs we now train the model using only 5 epochs on the combined training and validation data. To quickly evaluate the model we use the `evaluate` function that calculates the previously specified statistics, in this case loss and accuracy, for the provided data.

```
results <- model %>% evaluate(x_test, y_test)
```

We obtain a loss score of 0.327 and accuracy of 0.875. Rerunning the code will result in slight result deviation due to the stochastic nature of the neural network. To predict on new observations we run the following line,

```
predictions <- model %>% predict(x_test)
```

which provides a vector of probabilities for each observation, these are the probabilities used to produce the *receiver operating characteristic* (ROC) curve shown in Figure 5.5.

Trading Data Application

When applying neural networks to trading data we monitor the evolution of the accuracy and loss on both the training and validation data to get an idea of how many epochs are needed, just as we did above. We further look at the shape of the ROC-curve created from predictions on the validation set. Finally, we evaluate the profits we would make by trading according to the model. When applied to trading data, neural networks, which are variable by design, become even more variable, thus, to evaluate any model configuration we must run the code multiple times. We configure the model topography in an ad hoc fashion where we try different layer combinations with and without regularization while monitoring all the aforementioned performance statistics.

5.2 | Gradient Boosting

For gradient boosting we use the `xgboost` R-package, which we configure and train in the lines of code below. First we group the explanatory variables and labels for both the training and validation sets. We then train and configure the model. The max depth decides the depth of the boosted trees, which in turn also controls the model complexity. The `eta` argument is the learning rate which we can define anywhere in the interval $(0, 1]$, in this example we set it to 0.3. The number of threads, `nthreads`, is set to four and simply allows for parallel computations. The number of boosting rounds, `nrounds`, is 200 and we inform the model that we are performing a binary logistic regression. The `lambda` argument of zero stops the addition of ridge regularization, which we found to be invoked by default, even though the package documentation states otherwise.

```r
dtrain <- xgb.DMatrix(data = partial_x_train, label = partial_y_train)
dval <- xgb.DMatrix(data = x_val, label = y_val)
watchlist <- list(train = dtrain, test = dval)

model <- xgb.train(data = dtrain,
                   max_depth = 3,
                   eta = 0.3,
                   nthread = 4,
                   nrounds = 200,
                   watchlist = watchlist,
                   objective = "binary:logistic",
                   lambda = 0)
```

Since we supply both a training and a validation set we can extract the validation error as a function of the number of boosting iterations to check if we are overfitting. The following lines of code extract the training and validation errors shown in Figure 5.2.

```r
val_err <- data.frame(err = bst$evaluation_log$test_error)
val_err$iter <- 1:length(val_err$err)
train_err <- data.frame(err = bst$evaluation_log$train_error)
train_err$iter <- 1:length(train_err$err)
```

Since gradient boosting is fitting to adaptively reduce bias we need to ensure that we do not overfit, inspecting Figure 5.2 we see no evidence of overfitting. Since the training and validation errors do not seem to raise any concerns we fit the same model on the full dataset. The test error and accuracy from the model trained on the full dataset are extracted in the following lines of code, which result in an error of 0.138 and accuracy of 0.862.

```r
validation_probabilties <- predict(model, x_test)
validation_prediction <- (validation_probabilties > 0.5)
sum(validation_prediction == y_test)/length(y_test)
model$evaluation_log$test_error[200]
```

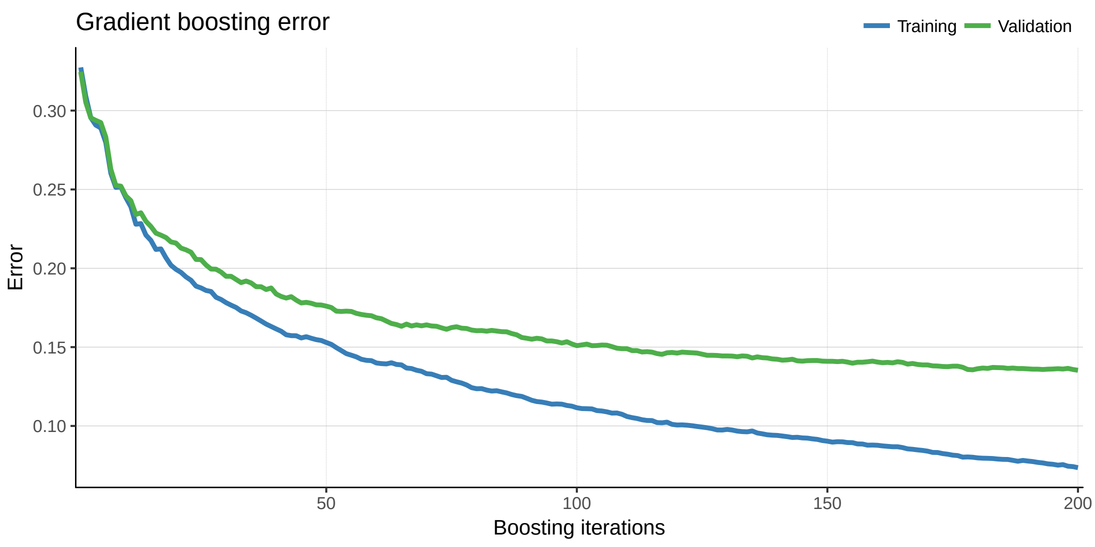


Figure 5.2: The training and validation errors, plotted against the number of boosting iterations, in the IMDb review classification example using gradient boosting.

Trading Data Application

When applying gradient boosting to trading data we monitor the evolution of the error on both the training and validation sets to ensure we use an adequate number of boosting iterations. As with neural networks we look at the shape of the ROC-curve, created from predictions on the validation set, and evaluate the profits. We decide the tree depth and learning rate in an ad hoc fashion where we try different combinations while monitoring all the aforementioned performance statistics.

5.3 | Random Forests

Since the random forests algorithm grows large de-correlated trees to reduce variance but not bias, as this should already be low, random forests can be trained without any configuration. We do not perform any configuration, thus, we train the model on the full dataset straight away. However, the default number of trees is 500, and with 10000 explanatory variables and a rather large number of observations this particular forest takes some time to grow. To fit the tree within reasonable time we limit the number of trees to 250 in this example, but for trading data we still use 500 trees.

```
model <- randomForest(x = x_train,
                      y = as.factor(y_train),
                      ytest = as.factor(y_test),
                      xtest = x_test,
                      do.trace = TRUE,
                      ntree = 250)
```

Random forests produce an OOB error estimate, which is plotted with the test error as a function of the number of trees in Figure 5.3. In this example we see that the OOB error estimate is consistently higher than the test error, which could be caused by the

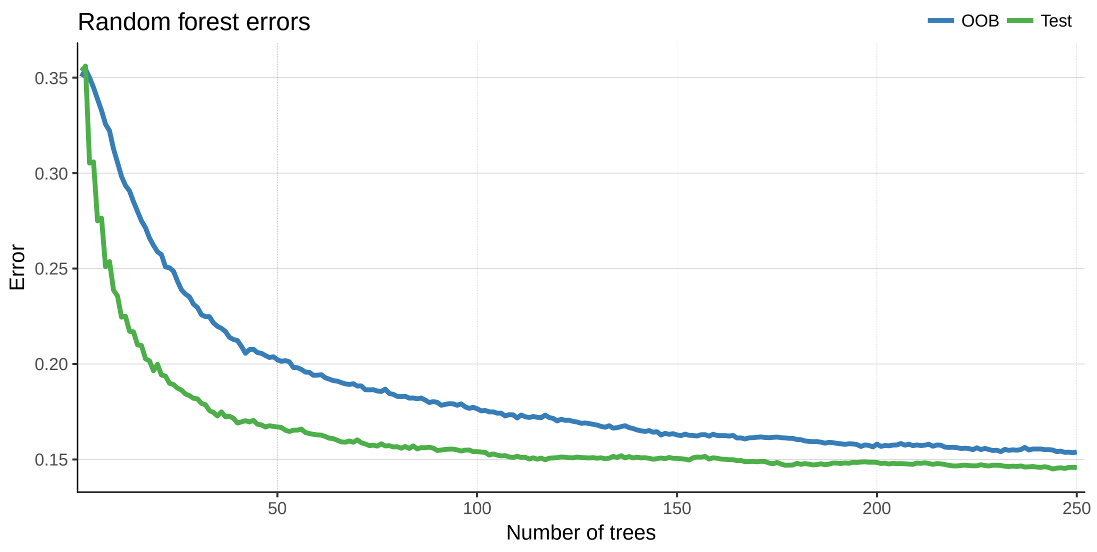


Figure 5.3: The OOB and test errors, plotted against the number of trees, in the IMDb review classification example using random forests.

OOB estimates being estimated from weaker models than the test errors. We note that the OOB and test errors seem to evolve in the same manner and furthermore seem to be converging. The test error and accuracy is extracted in the following lines of code, which result in a test error of 0.145 and accuracy of 0.854.

```
model$test$err.rate[250,1]
test_probabilities <- as.vector(model$test$votes[,2])
test_predictions <- test_probabilities > 0.5
sum(y_test == test_predictions)/length(y_test)
```

The application of random forests to trading data does not entail any additional steps since we do not perform any configuration.

5.4 | Threshold Analysis

As all considered models output prediction probabilities and use 50% as the threshold for classification, it might be worth considering how the classification changes subject to this treshold. To analyze the role of a classification threshold let us consider the four possible classification types that a binary classifier can produce, reported in Table 5.4. Our objective is to maximize the amount of true positives while minimizing the

		True class	
		Positive	Negative
Predicted class	Positive	True positive	False positive
	Negative	False negative	True negative

Figure 5.4: The possible types of predictions a binary classifier can produce.

amount of false positives, false positives are commonly referred to as type 1 errors. In the trading framework we do not concern ourselves much with false negatives (type 2 errors) as those correspond to missed investment opportunities, which is not as bad as type 1 errors that implies buying at undesirable times. To analyze the trade-off between true positives and type 1 errors we can use the ROC-curve.

To define the ROC-curve we first need to define the *true positive rate* (TPR), or *sensitivity*, and the *false positive rate* (FPR). Assume that data is given by $(y_i, x_i),\ i = 1, 2, \dots, N$ where $y_i \in \{0, 1\}$, and $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})$ is the explanatory variables. Further assume that we have N predicted probabilities given by

$$\pi(x_i) = P(y_i = 1 | X = x_i).$$

Define $T \in [0, 1]$ as the classification threshold and construct the classifier

$$\hat{y}_i = G(T; \pi(x_i)) = \mathbb{1}(\pi(x_i) \geq T), \tag{5.1}$$

that is, the predicted class of the i'th observation is $\hat{y}_i = 1$ if the estimated probability is larger than the threshold, T. As T varies from zero to one in (5.1) the number of true positives will decrease and so will the number of false positives, the ROC-curve is used to explore this dynamic. Let

$$\text{TPR} = \frac{\sum_{j:y_j=1} \mathbb{1}(y_j = \hat{y}_j)}{\sum_{i=1}^{N} \mathbb{1}(y_i = 1)},$$

which is the sum of all the predicted true positives, divided by the number of actual positives in the data. The FPR is defined as

$$\text{FPR} = \frac{\sum_{j:y_j=0} \mathbb{1}(\hat{y}_j = 1)}{\sum_{i=1}^{N} \mathbb{1}(y_i = 0)},$$

which is the sum of all false positives divided by the number of negatives in the dataset. Both TPR and FPR are functions of the chosen threshold, thus, we can vary T between one and zero, and for each value obtain a coordinate pair of TPR and FPR that when plotted makes up the ROC-curve. In Figure 5.5 we illustrate the trade-off between TPR and FPR as the classification threshold varies. On the curve itself the labelled points are the classification thresholds at that particular point.

Inspecting Figure 5.5 we see that we can potentially significantly reduce the number of false positives by increasing the threshold from 50% to 90%. In the trading framework reducing the number of false positives is desirable, especially for the creation of a risk averse trading strategy.

Note that the FPR can be defined as $1 - specificity$, where specificity is another term for the *true negative rate* (TNR) defined by

$$\text{TNR} = \frac{\sum_{j:y_j=0} \mathbb{1}(y_j = \hat{y}_j)}{\sum_{i=1}^{N} \mathbb{1}(y_i = 0)}.$$

Figure 5.5 also includes the *area under the curve* (AUC), which is a statistic that aids the interpretation of the ROC-curve. The AUC can be interpreted as the probability

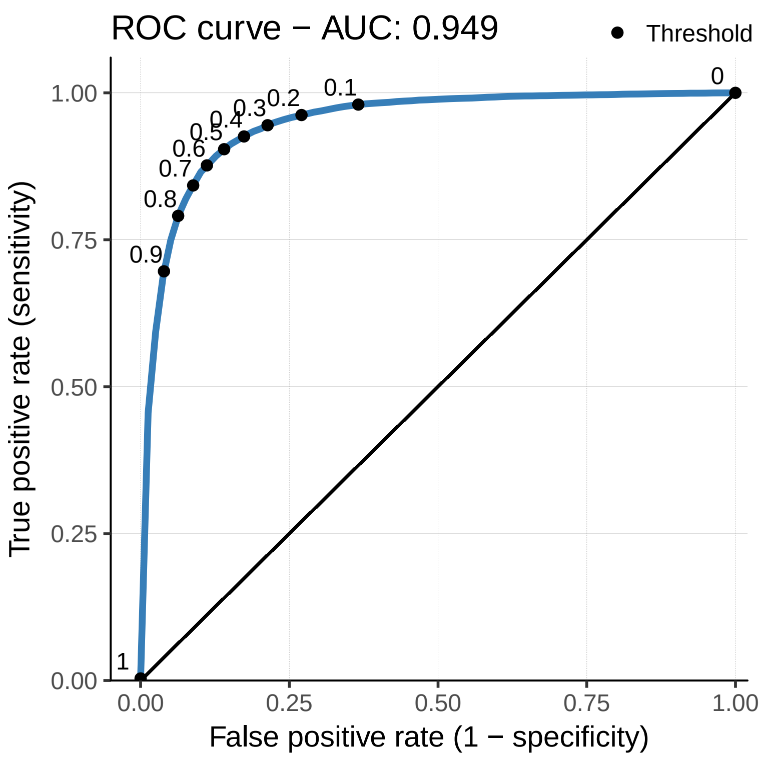


Figure 5.5: The ROC-curve generated from the IMDb review classification example using a neural network.

that a model assigns a higher probability to a randomly chosen positive observation than a randomly chosen negative one. The AUC can be used for model comparison, however, we choose to simply report it in the plots since the AUC can be a noisy statistic, which invalidates it as a consistent model comparison measure, see Hanczar et al. (2010) and Lobo et al. (2007).

Part II

Application

6 | Preliminary Study of BTC-USDT

In this chapter we perform preliminary tests of GLM, NN, GB, and RF on BTC-UDST trading data, using the data parameters described in the restricted setup in Section 1.5.3. Due to the amount of models included we do not perform an exhaustive search for the ideal data parametrization for each model, but follow the modelling procedure described in Section 6.1. In Section 6.2 we take a closer look at the BTC-USDT trading data. In Sections 6.3-6.5 we present the trading results of the GLM, NN, and tree based models, respectively. We finish the chapter by summarizing our findings in Section 6.6. For ease of discussion we sometimes refer to a model trained on, say, 30 minute candles as "30m model name", i.e., a neural network trained on 30m candles may be referred to as 30m NN.

6.1 | Modelling Procedure

Each of the models are trained on 15m, 30m, and 1h candles, where all numerical variables are scaled to have zero mean and unit variance. For each of the aggregation intervals we follow the greedy modelling procedure depicted in Figure 6.1. For each given model we first try to fit the model to the scaled data, and then try to improve the model by differencing, adding factors, and lagging. At each step we try to optimize the NN and GB configurations. Whether a certain step results in any improvement is measured by a combination of the shape of the ROC-curve, accuracy, and returns. When presenting the results we do not cover each step of the greedy modelling procedure, we only present the results from the final models for each aggregation interval.

6.2 | The BTC-USDT Trading Data

We consider BTC-USDT trading data aggregated into 15m, 30m, and 1h intervals in the period from February 11th, 2018 at 21:00 to May 1st, 2018 at 00:59. We partition the data into training, validation, and test sets, and report the the number of observations, buys, and stays in each set in Table 6.1. Throughout this chapter we only consider an initial guess for the limit and stop-limit percentages of 2% and 10%, respectively. The data partitioning corresponds to 60% of the data for training, 20% for validation and test each. The data partitioning is visualized in Figure 1.10. We note from Table 6.1 that as the aggregation interval increases the number of buys increases relative to the number of stays, and also slight deviations in the distribution of buys and stays among the sets. For the 15m candles the ratios of buys to stays are 0.74, 0.43, and 0.25 for the training, validation, and test sets, respectively. This change in distribution of buys and stays might be caused by some market sentiment changes that occur during the data period and might pose a challenge for the models.

The raw trading data contains the open, high, low, close, volume, and trades (number of trades), in Figure 6.2 we show the correlation between these variable for each

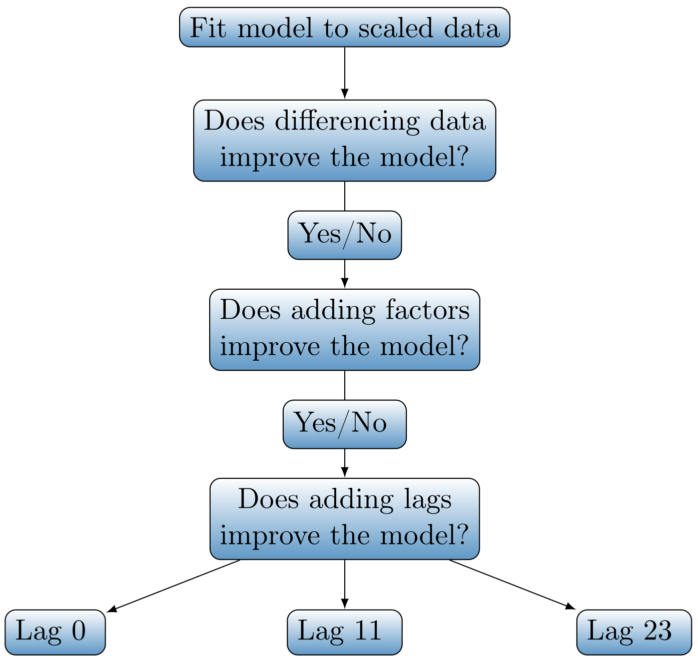

Fit model to scaled data
Does differencing data improve the model?
Yes/No
Does adding factors improve the model?
Yes/No
Does adding lags improve the model?
Lag 0
Lag 11
Lag 23

Figure 6.1: The greedy modelling procedure used for model parametrization and selection.

	15m			30m			1h		
	Total	Buys	Stays	Total	Buys	Stays	Total	Buys	Stays
Training	4475	1903	2572	2225	1266	959	1100	733	367
Validation	1494	450	1044	744	381	363	369	238	131
Test	1494	303	1191	744	299	445	369	222	147

Table 6.1: The number of observations, buys, and stays in the training, validation, and test sets for the BTC-USDT trading data aggregated into 15m, 30m, and 1h candles.

aggregation interval. We see that open, high, low, and close are practically perfectly correlated, which could imply that the explanatory power of the four might not be much higher than that of a single one. The open, high, low, and close correlations in Figure 6.2 are not exactly 1 but are rounded up from around 0.99. The high correlation is what motivates us to consider the first difference of these variables. We further note a high correlation between volume and number of trades, which is to be extected in some degree. Figure 6.3 shows the correlation on the same data but where the open, high, low, and close are differenced, yielding a much lower correlation that can perhaps aid the models. Figure 6.4 shows the correlation of the differenced variables and the added set of factors for the three aggregation intervals. We note that in Figure 6.4 the direction factor is highly correlated with the close.

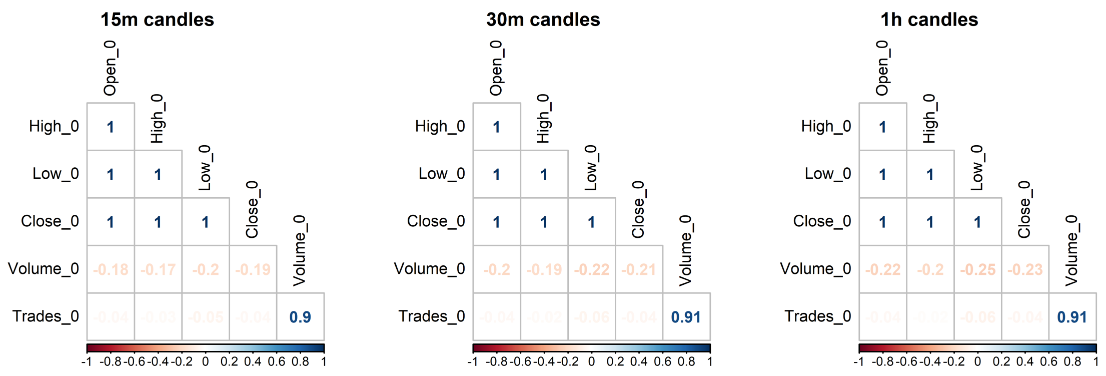


Figure 6.2: Correlation plot between the variables in the BTC-USDT trading data aggregated into 15m, 30m, and 1h candles.

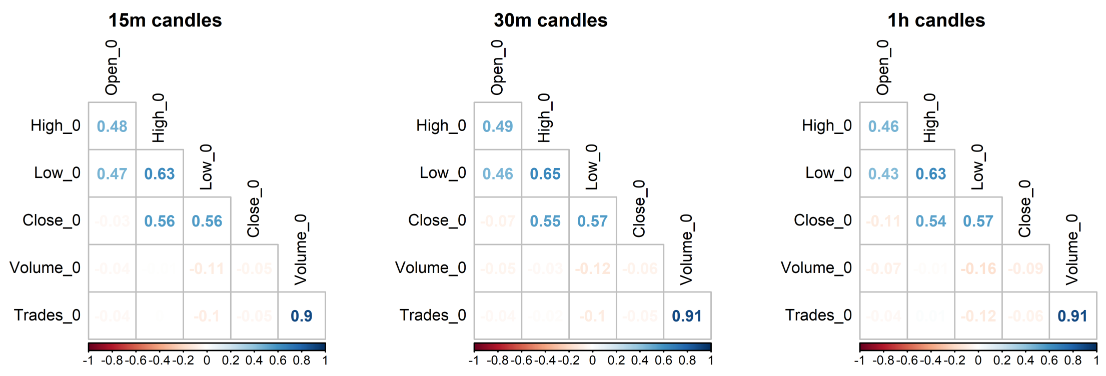


Figure 6.3: Correlation plot between the differenced variables in the BTC-USDT trading data aggregated into 15m, 30m, and 1h candles.

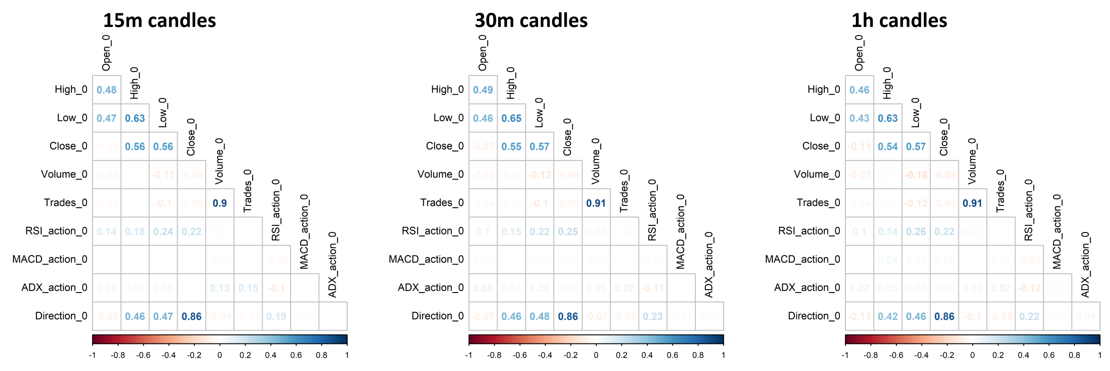


Figure 6.4: Correlation plot between the differenced variables in the BTC-USDT trading data aggregated into 15m, 30m, and 1h candles and derived factors.

6.3 | GLM Results

In this section we present the GLM trading results. To fit GLM we need to select λ and α values for the regularization described in Section 2.1.2. The λ value selected is the one that minimizes missclassification through cross-validaiton on the training set, and the α value selected is the one that maximizes returns on the validation set. Across all aggregation intervals a pure ridge regularization, $\alpha = 0$, seems to be the preferred choice by GLM. We find that for each aggregation interval there is some difference between the preferred data parametrizations. The preferred data parametrizations across the intervals are reported in Table 6.2.

Table 6.2 exhibits an interesting data parametrization pattern. For 15m candles lag 23 and factors are used, for 30m candles lag 23 is used but no factors, and for 1h candles lag 11 is used and no factors are preferred. It seems as the aggregation interval increases the preferred model becomes less complex, which might be a result of the reduction of observations. The validation and test results from trading based on GLM predictions

	Alpha	Difference	Lag	Factors
15m	0	0	23	Included
30m	0	0	23	Excluded
1h	0	0	11	Excluded

Table 6.2: The GLM configuration and data parametrization across models fitted on 15m, 30m, and 1h BTC-USDT candles.

are reported in Table 6.3. First thing we notice is that overall accuracy does not seem to be connected to returns. The percentage of buys that are true buys is 47%, 61%, and 66% for the 15m, 30m, and 1h candles, respectively. Thus, the percentage of true buys does not seem directly connected to returns either. The 15m GLM predicts 78 buys, of which 37 (47.4%) are true buys and 20 (25.6%) are losses, and yields a 49% profit. The 30m GLM predicts 173 buys, of which 106 (61.3%) are true buys and 49 (28.3%) are losses, and yields a 93% profit. The 1h GLM predicts 76 buys, of which 50 (65.8%) are true buys and 22 (28.9%) are losses, and yields a 51% profit.

The model using 30m candles performs best on the validation set, however, the models using 15m and 30m candles result in losses on the test set. The model using 1h candles is the only model able to produce profits on both the validation and test sets, with a 51% and 18% profit, respectively.

6.4 | Neural Network Results

In this section we present the trading results obtained by trading based on NN predictions. Generally neural networks seem to perform best when the open, high, low, and close variables are differenced and factors are added. In Table 6.4 we report the NN configurations and data parametrizations that produce the highest returns. Evaluating neural networks in this setup is very difficult since there is extremely high model variability, meaning that the same specification will produce very different results each

	Validation			Test		
	15m	30m	1h	15m	30m	1h
Buys	78	173	76	33	150	83
True buys	37	106	50	16	60	56
False buys	41	67	26	17	90	27
Stays	1416	571	293	1461	594	286
True stays	1003	296	105	1174	355	120
False stays	413	275	188	287	239	166
Losses	20	49	22	17	60	22
Accuracy	0.70	0.54	0.42	0.80	0.56	0.48
Fees	0.16	0.35	0.15	0.07	0.30	0.17
Return	0.49	0.93	0.51	-0.36	-0.39	0.18

Table 6.3: Trade summary on the validation and test sets using GLM for predicting trades on 15m, 30m, and 1h BTC-USDT candles.

time it is run. To account for the variability, and ensure that the improvements we see are not simply due to model variation, we have to run the model multiple times at each iteration of the modelling procedure. For all the neural networks we use a batch size of 512 as we are unable to obtain any discernible improvements by changing this. For all of the neural networks we weigh the classes according to their prevalence; say we have twice as many stays as buys, then the stays gets weighed 0.5 and buys 1. The weighing is done to keep the model from simply classifying all observations according to the majority class.

	Layers	Layer nodes	Batch size	Epochs	Difference	Lag	Factors
15m	2	32, 32	512	10	1	0	Included
30m	2	16, 16	512	5	1	0	Included
1h	2	16, 16	512	10	1	0	Included

Table 6.4: The neural network specifications we find perform the best on 15m, 30m, and 1h candles using differenced data and including factors.

In Table 6.5 we report a summary of the trades generated by the neural networks on the validation set. Since we are not able to obtain stable neural networks we evaluate the models 200 times and report averages with confidence intervals. The values in Table 6.5 have been rounded to integers, except for accuracy, fees, and return. The 15m NN predicts 374 buys, of which 137 (36.6%) are true buys and 145 (38.8%) are losses, and yields a 23% profit. The 30m NN predicts 256 buys, of which 133 (51.9%) are true buys and 86 (33.6%) are losses, and yields a 36% profit. The 1h NN predicts 134 buys, of which 84 (62.6%) are true buys and 38 (28.4%) are losses, and yields a 4% profit.

We further note that the 15m NN has a higher overall accuracy but still produces a smaller profit than the 30m NN. It seems that the 30m NN misses more potential trade opportunities but is more accurate in the buys it ends up predicting. Even with a higher accuracy, when it comes to true buys, the 1h NN only yields a very small profit, however, we should keep in mind that as the aggregation interval increases, the potential

	15m	30m	1h
Buys	374 (366-382)	256 (242-270)	134 (127-141)
True buys	137 (134-139)	133 (125-140)	84 (80-89)
False buys	237 (232-243)	123 (116-130)	50 (48-53)
Stays	1120 (1112-1128)	488 (474-502)	235 (228-242)
True stays	807 (801-812)	240 (233-247)	81 (78-83)
False stays	313 (311-316)	248 (241-256)	154 (149-158)
Losses	145 (142-148)	86 (82-91)	38 (36-40)
Accuracy	0.63 (0.63-0.63)	0.5 (0.5-0.5)	0.45 (0.44-0.45)
Fees	0.75 (0.73-0.76)	0.51 (0.48-0.54)	0.27 (0.26-0.28)
Return	0.23 (0.2-0.26)	0.36 (0.32-0.4)	0.04 (0-0.07)

Table 6.5: Trade summary on the validation set using neural networks, the summary is based on results from running the model 200 times and contains the mean value of each variable along with a 95% confidence interval. The reported values are rounded to integers, except for accuracy, fees, and return.

profit is limited since there are fewer trading opportunities. It seems the highest profit is obtained by trading based on the 30m NN.

In Table 6.6 we report the test set results. We see that the 15m NN performs much worse with lower overall accuracy and a 21% loss. The 30m NN does not change much, neither in terms of accuracy or profit. The 1h NN starts performing better and went from the lowest profit to the highest of 62%, which might be a result of simply allowing the model to train on more observations. So perhaps for the neural network a 1h aggregation interval, or even higher with a sufficient number of observations, might yield a better overall result. Even though it seems there are some profits to be made the instability of the models is cause for concern.

	15m	30m	1h
Buys	507 (500-514)	301 (291-312)	160 (155-165)
True buys	124 (122-126)	124 (120-129)	97 (94-100)
False buys	383 (378-388)	177 (171-183)	63 (61-65)
Stays	987 (980-994)	443 (432-453)	209 (204-214)
True stays	808 (803-813)	268 (262-274)	84 (82-86)
Fase stays	179 (177-181)	175 (170-179)	125 (122-128)
Losses	196 (193-199)	106 (102-109)	52 (51-54)
Accuracy	0.62 (0.62-0.63)	0.53 (0.52-0.53)	0.49 (0.49-0.49)
Fees	1.02 (1-1.03)	0.6 (0.58-0.62)	0.32 (0.31-0.33)
Return	-0.21 ((-0.24)-(-0.18))	0.34 (0.3-0.38)	0.62 (0.58-0.66)

Table 6.6: Trade summary on the test set using neural networks, the summary is based on results from running the model 200 times and contains the mean value of each variable along with a 95% confidence interval. The reported values are rounded to integers, except for accuracy, fees, and return.

6.5 | Tree Based Classifier Results

In this section we present the trading results from trading on the validation and test sets based on GB and RF. The configuration and data parametrization for GB across aggregation intervals are reported in Table 6.7.

	Max depth	Eta	Iterations	Difference	Lag	Factors
15m	4	0.1	10	0	0	Excluded
30m	4	0.3	40	0	0	Included
1h	4	0.3	10	0	0	Excluded

Table 6.7: The gradient boosting configurations and data parametrizations we find perform the best in predicting trades on 15m , 30m, and 1h candles.

For the 15m GB we find that a maximum tree depth of 4, a learning rate of 0.1, and 10 boosting iterations using candles without factors, differencing, or lagging seems to be preferred. For the 30m GB we find that a maximum tree depth of 4, a learning rate of 0.3, and 40 boosting iterations using candles with factors, and no differencing or lagging seems to be preferred. For the 1h GB we find that a maximum tree depth of 4, a learning rate of 0.3, and 10 boosting iterations and candles without factors, differencing, or lagging seems to be preferred. For the 15m GB we find that a slightly lower learning rate seems to be preferred, which might be caused by the 15m candles being noisier compared to the other intervals. The 30m GB is the only model that seems to benefit from the addition of factors, which might also be why we find that more boosting iterations are preferred for this model. None of the models seem to benefit from differencing or lagging.

The random forests are built without any configuration. In Table 6.8 we report the default growing configurations from the `randomForest` R-package. Trees are the number of trees grown, when growing a tree two predictive variables are chosen with replacement, and the minimum required number of observations in the terminal nodes is one. As for data parametrization, all of the random forests seem to prefer using candles without factors, differencing, or lagging.

Trees	Sampled variables	Node size	Difference	Lag	Factors
500	2	1	0	0	Excluded

Table 6.8: The default random forest configuration using raw trading data.

In Table 6.9 we report the trading results from trading based on the gradient boosting predictions in the validation and test sets. First thing to notice is that we are now seeing higher profits on the validation set compared to NN and GLM. The 15m GB predicts 667 buys, of which 230 (34.5%) are true buys and 234 (35%) are losses, and yields a 177% profit. The 30m GB predicts 506 buys, of which 282 (55.7%) are true buys and 143 (28%) are losses, and yields a 273% profit. The 1h GB predicts 298 buys, of which 204 (68.5%) are true buys and 60 (20%) are losses, and yields a 219% profit.

The ability to predict true buys seems to increase with the aggregation interval, which is not surprising since higher aggregation intervals should filter out noise in the

	Validation			Test		
	15m	30m	1h	15m	30m	1h
Buys	667	506	298	217	368	346
True buys	230	282	204	36	144	207
False buys	437	224	94	181	224	139
Stays	827	238	71	1277	376	23
True stays	607	139	37	1010	221	8
False stays	220	99	34	267	155	15
Losses	234	143	60	113	129	113
Accuracy	0.56	0.57	0.65	0.70	0.49	0.58
Fees	1.34	1.02	0.60	0.43	0.74	0.69
Return	1.77	2.73	2.19	-1.42	0.19	1.52

Table 6.9: Trade summary from trading based on the gradient boosting predictions in the validation and test sets using 15m, 30m, and 1h candles.

data. As with the previous models, the performance, in terms of profits, of the 15m and 30m GB drops when we start trading on the test set. The 1h GB has the best test set profit of 152%. The 15m GB has the highest overall accuracy on the test set but this accuracy mainly comes from the ability to predict stays, which is probably why we experience a drop in profits.

	Validation			Test		
	15m	30m	1h	15m	30m	1h
Buys	754	493	304	495	396	233
True buys	256	274	204	82	148	148
False buys	498	219	100	413	248	85
Stays	740	251	65	999	348	136
True stays	546	144	31	778	197	62
False stays	194	107	34	221	151	74
Losses	267	139	67	203	139	71
Accuracy	0.54	0.56	0.64	0.58	0.46	0.57
Fees	1.51	0.99	0.61	0.99	0.79	0.47
Return	1.82	2.72	1.75	-0.92	0.32	1.05

Table 6.10: Trade summary from trading based on the random forest predictions in the validation and test sets using 15m, 30m, and 1h candles.

In Table 6.10 we report the trading results from trading based on the random forest predictions in the validation and test sets. The 15m RF predicts 754 buys, of which 256 (34%) are true buys and 267 (35.5%) are losses, and yields a 182% profit. The 30m RF predicts 493 buys, of which 274 (55.6%) are true buys and 139 (28.2%) are losses, and yields a 272% profit. The 1h GB predicts 304 buys, of which 204 (67.1%) are true buys and 67 (22%) are losses, and yields a 175% profit. Generally, the results of trading based on random forests is similar to those obtained from trading based on gradient boosting. The profits are large on the validation set and 30m seems to be the ideal interval. We note that once again profits are smaller on the test set and the 15m RF results in a loss. Based on the test set the best aggregation interval is once again 1h with a profit of 105%, and overall accuracy does not seem related to the returns.

6.6 | Summary

Generally, the models agree that 30m candles are preferred on the validation set. However, models trained using 30m candles generally do not fare well on the test set. The models do not seem to care much for the factors we add, all except 15m GLM and 30m GB did not improve from the inclusion of factors. This might be a result of the naive approach taken, where all factors are included simultaneously, and perhaps adding factors one by one might yield some improvements. For the tree based models, the 1h candles result in high profits on both the validation and test sets. Furthermore, the general consensus among the models is that as the aggregation interval increases the rate of true buys increases as well, thus 1h candles might be the best choice. For the tree based models, removing any of the open, high, low, and close variables results in deterioration of model performance even though the models prefer these variables raw, where they are highly correlated.

The models, while agreeing in some aspects, are not all performing equally well. The NN models are actually not too far behind in terms of test set profits, but the low validation set profits and the instability of the models seem to indicate that they are not fit for trading, atleast in this setting. The GLM models have a reasonable performance on the validation set but yield negative returns on both 15m and 30m candles on the test set, with a small profit of 18% on the 1h candles. The GB and RF models seem to be the best performing models on the validation set, but still yields a loss on the 15m candles in the test set and a low profit on the 30m candles. However, the 1h GB and RF models do yield high profits on both validation and test sets.

The experimental setup in this chapter, while giving us a feel for what works and what does not, is not the ideal setup to use for trading. In the current setup we train the models on a fixed period and proceed to predict trades in a future period. In terms of Hypothesis 1, we assume that what we are actually estimating, is some time dependent function f_t subject to change during the prediction period. To test whether this is the case we can perform a rolling training and prediction procedure summarized by repeating the following steps.

1. Train the model on all available data up to the current point in time.
2. Once a new candle is available, predict whether it is a buy or stay and trade accordingly.
3. Add the newly received candle to the training data, drop the oldest candle, and retrain the model.

The magnitude of profits that we attempt to predict might also influence the results, predicting 2% is simply a choice we make. Predicting other magnitudes of profits, considering different aggregation intervals and profit horizons might also improve the results.

7 | Model Improvement

In Chapter 6 we perform a preliminary study of the performance of GLM, NN, GB, and RF on the BTC-USDT trading data. In this chapter, guided by the results in Chapter 6, we narrow down the field of candidate models and perform a more exhaustive modelling procedure. So far we find evidence that the models perform better on 1h candles overall, so we only consider improving models on this interval. Furthermore, GB and RF seem to clearly outperform GLM and NN, while being stable as well, thus, we only consider the tree based models.

7.1 | Profit Parametrization

As we mention in Section 6.2, trying to classify buys with a 2% limit and 10% stop-limit is an initial guess and a parametrization of the dataset. The stop-limit of 10% effectively corresponds to no stop-limit since it is unlikely to trigger during the 24 candle horizon we use. In this section we investigate whether we can improve the profits by changing the limit and stop-limit parametrizations of the data. To investigate different limits and stop-limits we vary the limit and stop-limit, and calculate the potential profits for each combination. Potential profits are calculated as the profits made if we correctly classify every candle in the set. Potential profits are calculated on the combined training and validation set, which is a long period, making the profits significantly higher than anything we report in Chapter 6.

We search across a grid where limit $= \{0.01, 0.02, 0.03, 0.04, 0.05\}$ and stop-limit $= \{0.05, 0.06, 0.07, 0.08, 0.09, 0.1\}$. For a 2% limit and a 10% stop-limit the potential profit is 1745% over the combined training and validation set. The highest potential profit is obtained by setting the limit to 4% and the stop-limit to either 9% or 10%, which both yield a potential profit of 2255%. The stop-limits of 9% or 10% produces the same profit because neither triggers, we proceed using the lowest stop-limit of 9%.

7.1.1 | Model Configurations

Initially we use the configurations of Section 6.5 and gradually try to improve by monitoring changes in returns. Differencing the data does not seem to improve the tree based models so we proceed only using undifferenced data. We add factors one by one and test the different combinations before we add lags to the models. As described in Section 7.1, we also change the limit to 4% and stop-limit to 9%, the potential best combination. Model performance is still only evaluated on the validation set.

GB seems to benefit from the addition of class weights where the majority class is down-weighed. The weights are calculated as

$$\begin{aligned}\text{Weight buy} &= \frac{\sum_{i=1}^{N} \mathbb{1}(y_i = 1)}{N},\\ \text{Weight stay} &= \frac{\sum_{i=1}^{N} \mathbb{1}(y_i = 0)}{N},\end{aligned}$$

where N is the number of observations, buys are encoded $y_i = 1$, and stays as $y_i = 0$. GB does not improve further from the addition of any of the factors or lags used. GB does seem to prefer a higher number of boosting iterations in this setting, compared to what we observe in Section 6.5. The configuration and data parametrization for GB are reported in Table 7.1.

Max depth	Eta	Iterations	Lag	Factors	Class weights
4	0.3	60	0	Excluded	Included

Table 7.1: The gradient boosting configuration and data parametrization we find performs best in predicting trades on 1h candles, using a limit of 4% and stop-limit of 9%.

For the RF we use the same configuration for training, as we do in Section 6.5, and as for data parametrization we see improvements from the addition of hours as factor and the RSI trading signal, described in Appendix A. The RF configuration and data parametrization are reported i Table 7.2.

Trees	Sampled variables	Node size	Lag	Factors	Class weights
500	2	1	0	Hours, RSI	Excluded

Table 7.2: The random forests configuration and data parametrization we find performs best in predicting trades on 1h candles, using a limit of 4% and stop-limit of 9%.

7.1.2 | Returns

In Table 7.3 we report the trading results on the validation and test sets from trading based on both GB and RF. On the validation set both models exhibit some profit improvements. GB predicts 175 buys, of which 71 (40.6%) are true buys and 41 (23.4%) are losses, and yields a 250% profit. RF predicts 183 buys, of which 70 (38.3%) are true buys and 41 (22.4%) are losses, and yields a 264% profit. Interestingly, RF yields a higher profit with a lower buy accuracy than GB, which is probably because the false buys RF produces are not as bad as those of GB.

Unfortunately both buy accuracy and profit drop as we trade on the test set. GB predicts 86 buys, of which 26 (30.2%) are true buys and 39 (45.3%) are losses, and yields a 34% profit. RF predicts 71 buys, of which 23 (32.4%) are true buys and 31 (43.7%) are losses, and yields an 18% profit. Even though we can improve the validation set profits our findings do not generalize well on the test set. Recall that in Section 6.5 we obtain 152% and 105% profits for GB and RF, respectively. Since the model, despite the higher validation profits, does not generalize well to the test set we conclude that using a 4% limit and 9% stop-limit does not seem to improve over the initial guess, of a 2% limit and 10% stop-limit.

	Validation		Test	
	GB	RF	GB	RF
Buys	175	183	86	71
True buys	71	70	26	23
False buys	104	113	60	48
Stays	194	186	283	298
True stays	137	128	192	204
False stays	57	58	91	94
Losses	41	41	39	31
Accuracy	0.56	0.54	0.59	0.62
Fees	0.35	0.37	0.17	0.14
Return	2.50	2.64	0.34	0.18

Table 7.3: Trade summary from trading based on the GB and RF predictions in the validation and test sets using 1h candles with a 4% limit and 9% stop-limit.

7.2 | Further Model Calibration

Since we are not able to improve the models by using different limits and stop-limits we now recalibrate the 1h GB and RF presented in Chapter 6, i.e., the models trained on data classified using a 2% limit and 10% stop-limit. The factors added in Chapter 6 are all added at once, and if no improvement is evident they are excluded from the models. In this section we try to add the factors one by one and test different combinations of the factors. Furthermore, as we see benefits from adding weights to the GB in Section 7.1, we also test whether the same improvements are possible in this setup. Model performance is still only evaluated on the validation set.

We are able to increase the validation set profits for GB by adding class weights, as in Section 7.1, and find no evidence that it would be beneficial to change the model configuration reported in Table 6.7. RF can be improved by the addition of the MACD and ADX trading signals. The addition of class weights has no effect on RF and the configuration is the same as reported in Table 6.8.

The trading results of the improved models on the validation and test sets are reported in Table 7.4. GB now has a 234% profit on the validation set, which is a 15% increase. RF now has a 195% profit on the validation set, which is a 20% increase. Unfortunately, the improvements seen on the validation set for GB do not generalize well to the test set where GB yields an 18% profit. However, RF does generalize well to the test set and yields a 111% profit, which is a 6% increase. It seems we successfully improve RF, but can not improve GB beyond the naive model presented in Section 6.5.

	Validation		Test	
	GB	RF	GB	RF
Buys	271	309	45	235
True buys	191	210	30	150
False buys	80	99	15	85
Stays	98	60	324	134
True stays	51	32	132	62
False stays	47	28	192	72
Losses	50	65	14	70
Accuracy	0.66	0.66	0.44	0.57
Fees	0.54	0.62	0.09	0.47
Return	2.34	1.95	0.18	1.11

Table 7.4: Trade summary from trading based on the further calibrated GB and RF predictions in the validation and test sets using 1h candles with a 2% limit and 10% stop-limit.

7.3 | Rolling Classification

In Section 6.6 we mention that it might not be ideal to only train the model once and use it to predict through the whole test set. The motivation being that we suspect the models can pick up some local market dynamics, which is specific to the prediction period under consideration. In Hypothesis 1 we assume the existence of some time dependent function that has the ability to predict profits. This time dependent function could change during the periods of the validation and test sets, causing a deterioration in the predictive ability. In this section we train the model on the training and validation sets, and perform the rolling classification described in Algorithm 4 on the test set. In Algorithm 4, N is the total number of observations in the combined training and validation set, henceforth referred to as the combined training set, and $\bar{N}$ is the number of observations in the test set. The algorithm starts by training the model on the combined training set, and then classifies the first observation in the test set. After each classification the observation we classified is then added to the combined training set and the oldest observation dropped. Algorithm 4, like the previous classification method, produces a vector of buys and stays of length $\bar{N}$. We then proceed to evaluate this vector of classifications in the same manner as we do with the previous classification method. We use the RF model presented in Section 7.2 and the GB model presented in Section 6.5. Additionally, we create an ensemble (Ens) model that only trades when both GB and RF agree on a buy. The trading results are presented in Table 7.5. GB and RF yield profits of 202% and 160%, which is an increase of 50% and 49%, respectively. The overall accuracy of both GB and RF also increases and the percentage of true buys is much closer to the overall accuracy. Ens arrives at a slightly lower profit of 153%, naturally it performs fewer trades than GB and RF which could result in a lower liquidity requirement.

To further examine whether the improvements seen are actually results of an increased ability to pick up local market dynamics, we perform an expanding classifi-

Algorithm 4: Rolling Classification

```
1 - Define a training set $(y_i, x_i),\ i \in \{1, 2, \ldots, N\}$ and test set
    $(y_j, x_j),\ j \in \{N+1, N+2, \ldots, N+\bar{N}\}$.
2 - for $l \in \{0, 1, \ldots, \bar{N}-1\}$
      i - Train the model on $(y_\tau, x_\tau), \tau \in \{1+l, 2+l, \ldots, n+l\}$.
     ii - Classify observation $(y_{n+l+1}, x_{n+l+1})$.
    iii - Store classification.
```

cation, i.e., we do not drop old observations as new ones are included. In terms of Algorithm 4, this corresponds to defining $(y_\tau, x_\tau), \tau \in \{1, 2, \ldots, n+l\}$ as the combined training set. We do not show trade summaries for this setup but profits decrease to 141%, 132%, and 112% for GB, RF, and Ens, respectively, which further supports the importance of the local market dynamics.

	GB	RF	Ens
Buys	297	240	222
True buys	199	166	156
False buys	98	74	66
Stays	72	129	147
True stays	49	73	81
False stays	23	56	66
Losses	76	62	54
Accuracy	0.67	0.65	0.64
Fees	0.60	0.48	0.45
Return	2.02	1.60	1.53

Table 7.5: Trade summary using GB, RF, and Ens to perform a rolling classification of trades on 1h candles with a 2% limit and 10% stop-limit on the test set.

7.3.1 | Further Examination of the Local Market Dynamics Hypothesis

In Section 7.3 we show that when performing rolling classification, the inclusion of older observations causes profits to deteriorate. Thus we see evidence that the function, f_t, in Hypothesis 1 seems to change over the course of the 369 observations in the test set. Now, if f_t changes over 369 observations it likely changes even more over the 1469 observations in the combined training set. In this section we investigate how profits change as we gradually reduce the size of the combined training set by excluding the oldest observations. We exclude observations in weekly increments, train the model, and calculate profits by rolling classification.

Figure 7.1 shows the evolution of profits as we reduce the combined training set. It seems that in general, after an initial dip, the profits start to increase as the combined training set is further reduced. The ensemble model seems to mainly follow the evolution

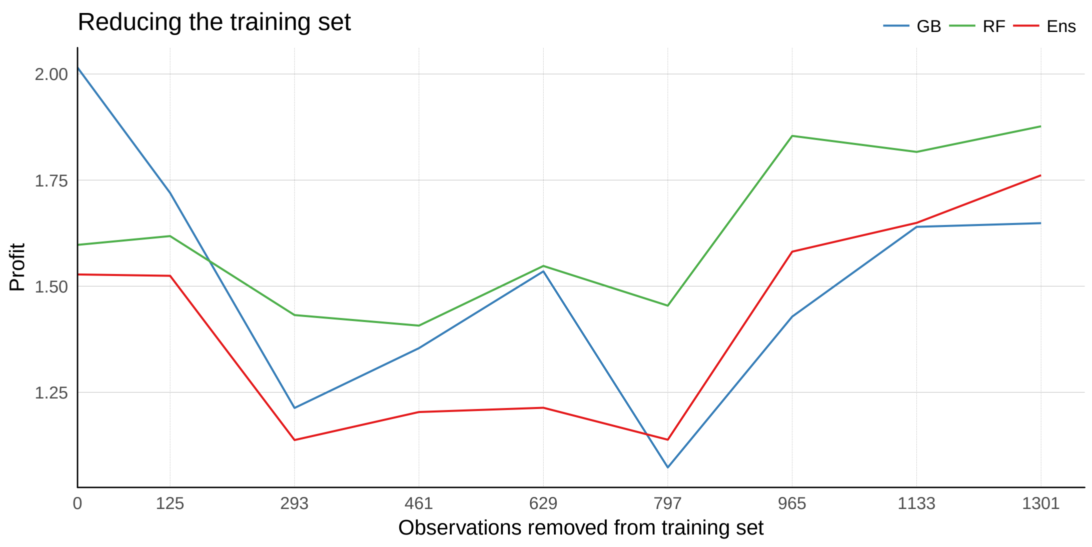


Figure 7.1: The evolution of profits calculated by a rolling classification, as the size of the training set is reduced by removing the oldest observations in weekly increments.

of RF. At the last observation in Figure 7.1 we are only training the models on a single week of data, 168 observations.

We see clear evidence that RF benefits from the removal of training observations. GB, however, does not show the same clear pattern, we see some of the same movements but the final profits are lower than those obtained by using the full combined training set. Perhaps further improvements can be obtained by further reducing the training set, however, we do not pursue these.

In Table 7.6 we report the trade summary produced by performing a rolling classification on the test set, where the models are trained on the 168 observations preceding the observations they are predicting. We see that RF improves both in terms of ac-

	GB	RF	Ens
Buys	216	218	192
True buys	155	161	147
False buys	61	57	45
Stays	153	151	177
True stays	86	90	102
False stays	67	61	75
Losses	50	47	39
Accuracy	0.65	0.68	0.67
Fees	0.43	0.44	0.39
Profit	1.65	1.88	1.76

Table 7.6: Trade summary using GB, RF, and Ens to perform a rolling classification of trades on 1h candles with a 2% limit and 10% stop-limit on the test set. The models are trained on a reduced combined training set containing only 168 observations.

curacy and profits, while GB does seem to experience a rather large profit decrease. Interestingly, the ensemble model improves in terms of accuracy and profit even though the GB profit decreases. The GB profit decrease might be caused by the fact that we still use the GB configuration derived in Chapter 6, which is a vastly different setup compared to the current. Thus, GB might benefit from a reconfiguration in the new setup, however, we do not pursue this. It is worth noting that GB actually increases in terms of true buy accuracy, from 67% to 71.8%, which is probably why the ensemble model improves as well.

Since RF improves overall by the reduction of the training set it seems clear that using 168 observations for training is the ideal choice in this case. For GB the conclusion is slightly more tricky since the profits decrease, however, GB benefits from the usage of a rolling classification, supporting the hypothesis that f_t changes over a short period of time. Furthermore, GB improves in terms of true buy accuracy from the reduction of the training set. Thus, we believe that reducing the training set to 168 observations is also the optimal choice for GB.

8 | Model Evaluation

In Chapter 7 we find that, for predicting trades on 1h candles using a 2% limit and 10% stop-limit, the best models are GB and RF using the configurations derived in Sections 6.5 and 7.2, respectively. For convenience, the optimal configurations and data parametrizations we find for GB and RF are restated in Tables 8.1 and 8.2, respectively. In this chapter we train the derived models on the combined training set and further evaluate their performance on the test set, where we also include the ensemble. Additionally, since we have (ab)used the test set for inference during some of the model selection steps we evaluate the models on a new BTC-USDT dataset. We further evaluate the models on data from the other cryptocurrency pairs discussed in Section 1.5.3: ETH-USDT, BNB-USDT, NEO-USDT, LTC-USDT, and BCC-USDT.

Max depth	Eta	Iterations	Factors	Weights
4	0.3	10	Excluded	Excluded

Table 8.1: The GB configuration and data parametrization we find performs the best in predicting trades on 1h candles using a 2% limit and 10% stop-limit.

Trees	Sampled variables	Node size	Factors	Weights
500	2	1	MACD, ADX	Excluded

Table 8.2: The RF configuration and data parametrization we find performs the best in predicting trades on 1h candles using a 2% limit and 10% stop-limit.

In Figure 8.1 we show the ROC-curves from the probabilities generated through the rolling classification on the test set. The high variability in data and the difficulty of the classification problem is clear by comparing these ROC-curves to the one from the IMDb example in Figure 5.5. The ROC-curve for RF seems slightly smoother and also has a higher AUC than that of GB. From Figure 5.5 we do not see any obvious improvements to be made from changing the threshold. However, RF ROC-curve does seem slightly interesting once the threshold exceeds 0.9.

Figure 8.2 shows the average relative importance plots for GB and RF obtained through the rolling classification on the test set. The importance is measured as mean Gini decrease, which is the average decrease of impurity obtained by using a certain variable for splitting. We train the models 369 times during the rolling classification where the importance of the variables changes each time, thus, the plots in Figure 8.2 are calculated as averages over the 369 models trained. Considering the importance plot for GB, we see that the closing price is the most important variable. This is very interesting due to the high correlation between open, high, low, and close, which could cause GB to be indifferent between the four. On the RF importance plot we see a more equally distributed importance, where close is still the most important variable. This highlights the difference between the two models as GB can choose from all six included

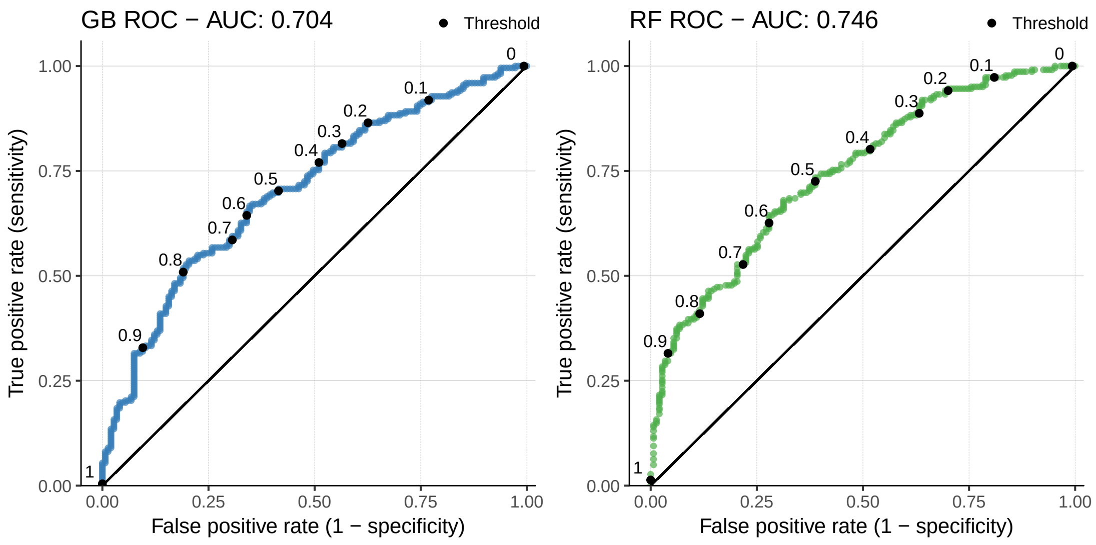


Figure 8.1: ROC-curves based on the GB and RF probabilities obtained through the rolling classification on the test set in the period from April 15th, 2018 at 16:00 to May 1st, 2018 at 00:59, performed in Section 7.3.1.

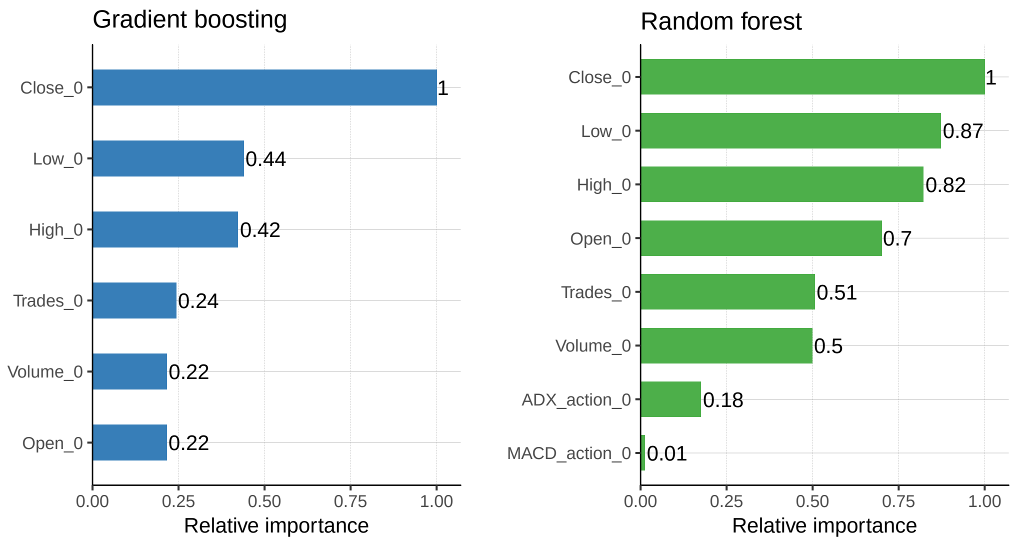


Figure 8.2: Average feature importance for the GB and RF obtained through the rolling classification on the test set in the period from April 15th, 2018 at 16:00 to May 1st, 2018 at 00:59, performed in Section 7.3.1.

variables on each split when growing tress and repeatedly chooses open, whereas RF can choose only between two randomly selected variables at each split. The RF importance plot indicates that despite the high correlation, open, high, low, and close generally do not have the same predictive power. The importance does seem to indicate that low and high have somewhat similar predictive power. While inspecting the BTC-USDT data we also see a high correlation between trades and volume, these variables, as opposed to open, high, low, and close, actually seem to have roughly the same predictive power. RF also uses the ADX factor which seems to be somewhat important, atleast compared to the MACD factor which could probably be dropped from the model without loosing much predictive power.

Figure 8.3 shows the trade performance of GB, RF, and Ens on the test set. In the top plot we see the 1h candles plotted over the test period with an upwards going price trend, which intuitively should make it easier for the models to yield profits. The middle plot depicts when each of the models chooses to buy, true buys yielding a 2% profit are coloured blue, false buys yielding a profit are green, and false buys yielding a loss are red. The bottom plot shows the cumulative returns of the models during the test period. Generally we see a very similar performance across all models. Considering the period at the start of the plot between April 15th and 18th, we note that the period starts out with a dip where all models avoid trading, except a single trade by GB. Then after the dip we see an upward price trend in which most of the total profits are made for all three models. After the peak on April 25th the models make a few bad trades but slowly make up for it throughout the remainder of the test period. As we mention, the upward price trend might make trading easier for the models, thus, it would be interesting to see how the models fare in a period with a decreasing price trend.

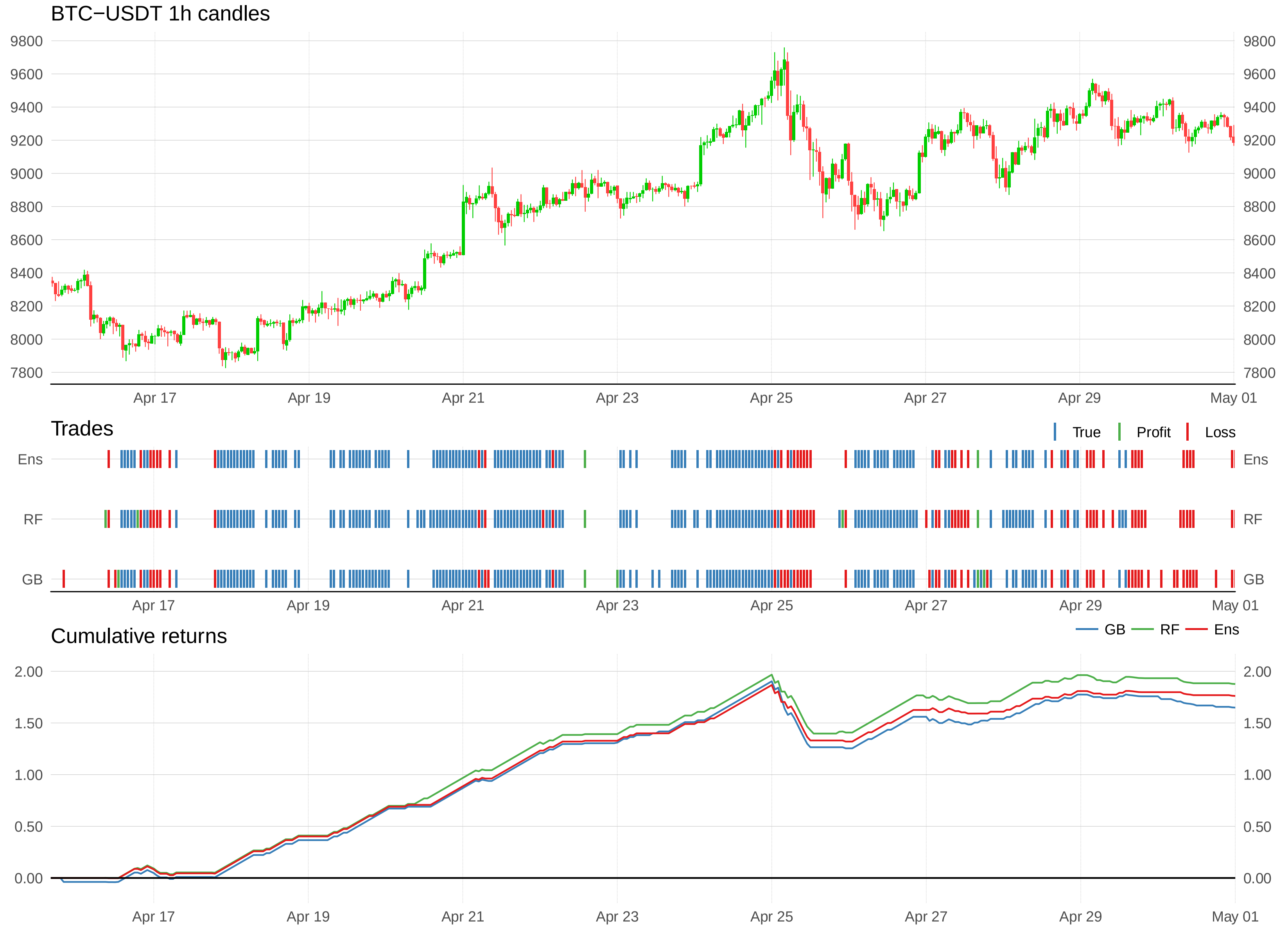


Figure 8.3: Model performance on the BTC-USDT 1h candles in the period from April 15th, 2018 at 16:00 to May 1st, 2018 at 00:59. **Top:** The candles in the period. **Middle:** The models' classifications of buys and stays. True (blue) are correctly classified buys resulting in a 2% profit, Profit (green) are wrongly classified buys that resulted in a profit, and Loss (red) are wrongly classified buys resulting in a loss. **Bottom:** The cumulative returns of the models through the period.

8.1 | New Data

To asses the performance on a "true" test set we obtain a new test set containing 1h BTC-USDT candles in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59. From here, when we refer to "test set" we are referring to the new test set, and when we compare to the test set in the previous section we refer to it as the old test. We perform the same trading based on rolling classification as on the old test set, and asses the results throughout this section.

In Table 8.3 we report the trade summary from trading on the test set. We note that the accuracy of all models increase compared to what we see on the old test set. The increased overall accuracy unfortunately does not translate to increased profits as we see a steady decrease in profits across the models. Overall, the true buy accuracy has decreased, thus, the increase in overall accuracy is obtained through an increased ability to predict stays.

	GB	RF	Ens
Buys	162	168	149
True buys	109	117	107
False buys	53	51	42
Stays	207	201	220
True stays	158	160	169
False stays	49	41	51
Losses	29	23	19
Accuracy	0.72	0.75	0.75
Fees	0.33	0.34	0.30
Return	1.00	1.42	1.28

Table 8.3: Trade summary using GB, RF, and Ens to perform a rolling classification of trades on 1h candles with a 2% limit and 10% stop-limit on the test set in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59.

The increase in overall accuracy does, however, translate into some better looking ROC-curves as shown in 8.4. The curves seem to be slightly smoother and the AUC has increased for both curves. We still do not see any obvious improvements to be had from changing the threshold, but note the same interesting behaviour from the RF ROC-curve when the threshold exceeds 0.9.

The average relative importance is obtained in the same manner as in Figure 8.2 and shown in Figure 8.5. We see that close is still the variable with the highest predictive power. For both models, the importance of high has increased, and from the RF importance plot we see that the predictive power of high is now very similar to that of close. For GB, trades and volume no longer have a similar importance, and volume is now the least important variable. For RF we see that the MACD factor still does not seem to add much to model performance, and furthermore, the importance of the ADX factor has also decreased. Figure 8.6 shows the trade performance of GB, RF, and Ens on the test set. First we note that now the price follows an overall downwards trend, which is an explanation for the decrease in profits. The test set starts out with a

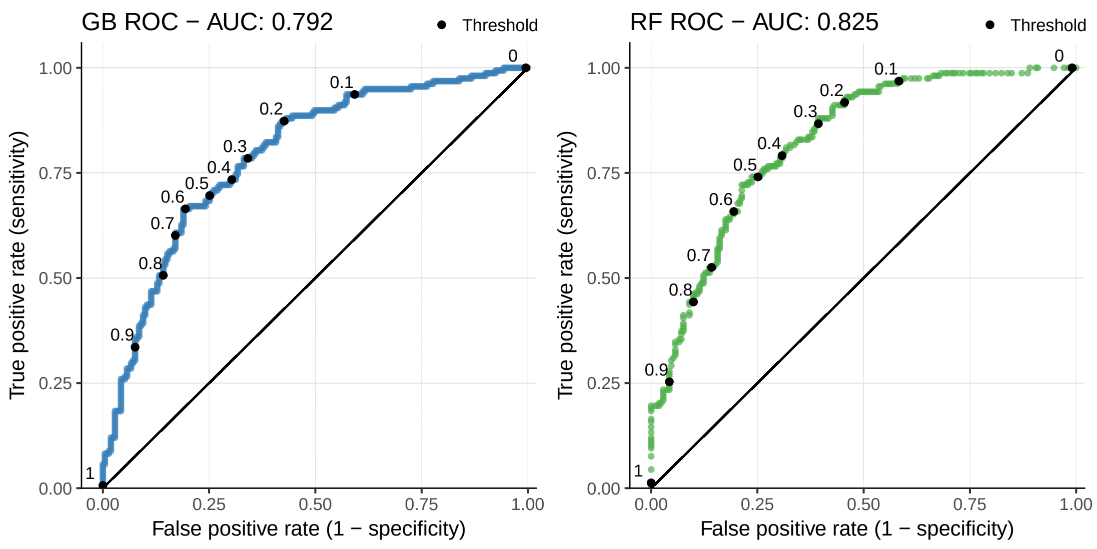


Figure 8.4: ROC-curves based on the GB and RF probabilities obtained through the rolling classification on the test set in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59.

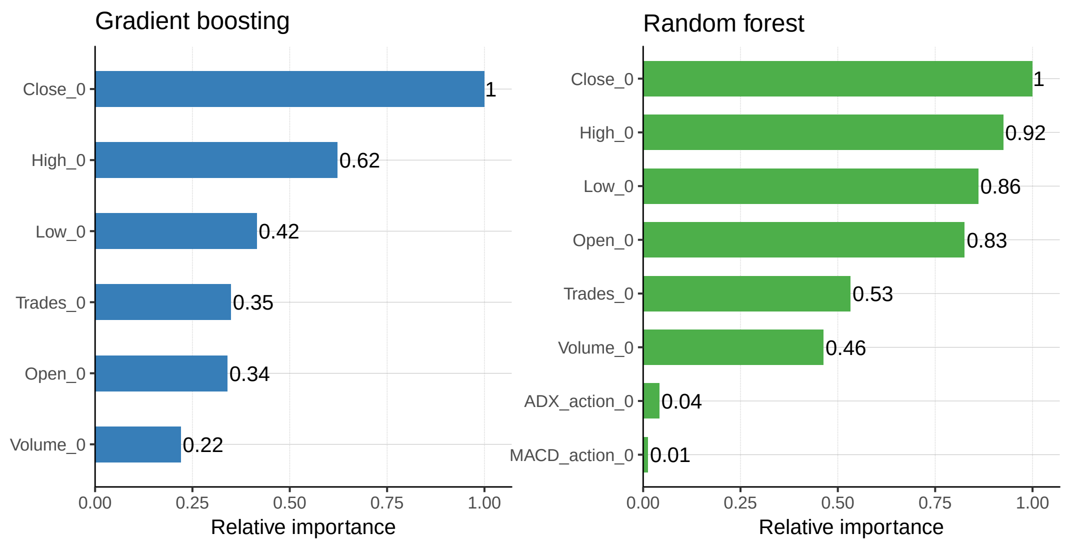


Figure 8.5: Average feature importance for the GB and RF obtained through the rolling classification on the test set in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59.

slight decrease, which none of the models trade on, and then proceeds to increase until a peak is reached at around May 6th, which is where the majority of profits are made. However, even though the price shows a steady decrease from May 6th to the end of the set, all models still manage to make more profits. GB is performing the worst and RF the best. The models exhibit the same patterns overall.

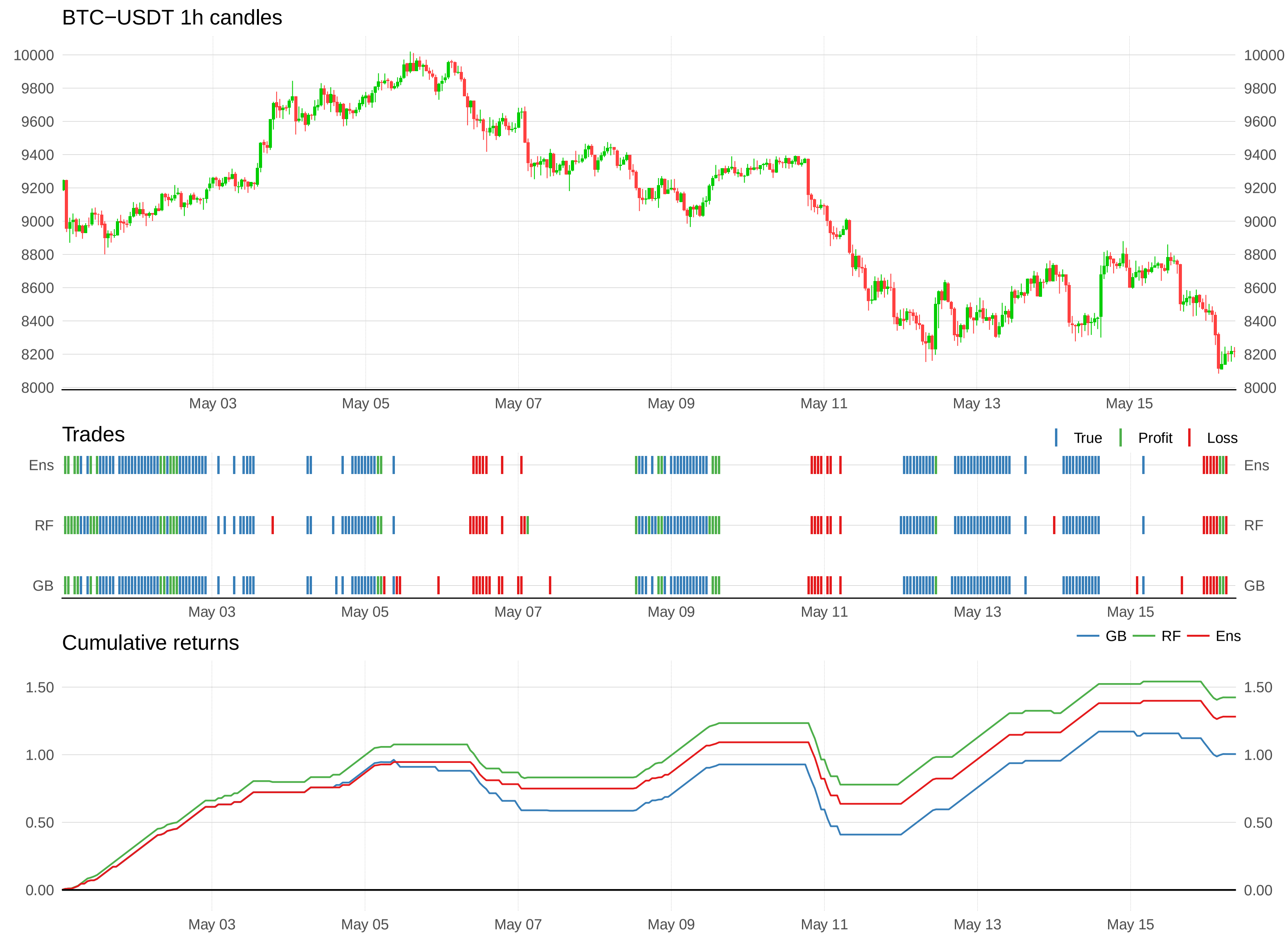


Figure 8.6: Model performance on the BTC-USDT 1h candles in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59. **Top:** The candles in the period. **Middle:** The models' classifications of buys and stays. True (blue) are correctly classified buys resulting in a 2% profit, Profit (green) are wrongly classified buys that resulted in a profit, and Loss (red) are wrongly classified buys resulting in a loss. **Bottom:** The cumulative returns of the models through the period.

8.2 | Other Pairs

In this section we explore how GB, RF, and Ens perform on ETH-USDT, BNB-USDT, NEO-USDT, LTC-USDT, and BCC-USDT compared to BTC-USDT. Since all pairs trade against USDT we drop "-USDT" when discussing the pairs. We use the data parametrization and model configurations derived for the BTC pair for the other trading pairs. For completion we report the trade summaries for all six pairs in Table 8.10 and the trading performance plots of the five other trading pairs are found in Appendix C.

The cumulative returns for the models on all six pairs are shown in Figure 8.7. All models are able to make a profit on five of the six pairs, the only pair on which the models are yielding losses is the BCC pair, with a 101%, 96%, and 87% loss for GB, RF, and Ens, respectively. Interestingly, BCC is the pair that the models made the highest profit on overall until around May 8th after which it goes down. Since we derive the data parametrization and model configurations using BTC we would expect BTC to be most profitable pair, this is however not the case. GB and Ens both make higher profits on BNB, of 138% and 153%, which is 38% and 25% more than on BTC. RF performs about the same on BTC and BNB, only making a 3% higher profit on BNB. Generally it seems like Ens is the model with the best performance, RF seems to come in second, and GB seems to have the lowest overall performance.

To figure out exactly which model performs the best across all six pairs we show the cumulative returns aggregated over the six pairs in Figure 8.8. RF seems to be making slightly higher profit on the first half of the test period, but after that Ens takes the lead and ends up outperforming both GB and RF. The fact that Ens manages to outperform GB and RF is an interesting result, it implies that the two models make different types of mistakes but agree on the true buys.

In Figure 8.9 we show the final return for the six pairs. We see that for all pairs, except for BTC, the Ens yields the highest return, thus, combining GB and RF has its merit. The magnitude of improvement in Ens compared to GB and RF varies across the pairs and is most pronounced in the NEO pair where Ens makes a profit of 73%, 9% higher than the combined profit of GB and RF, which make 25% and 39%, respectively.

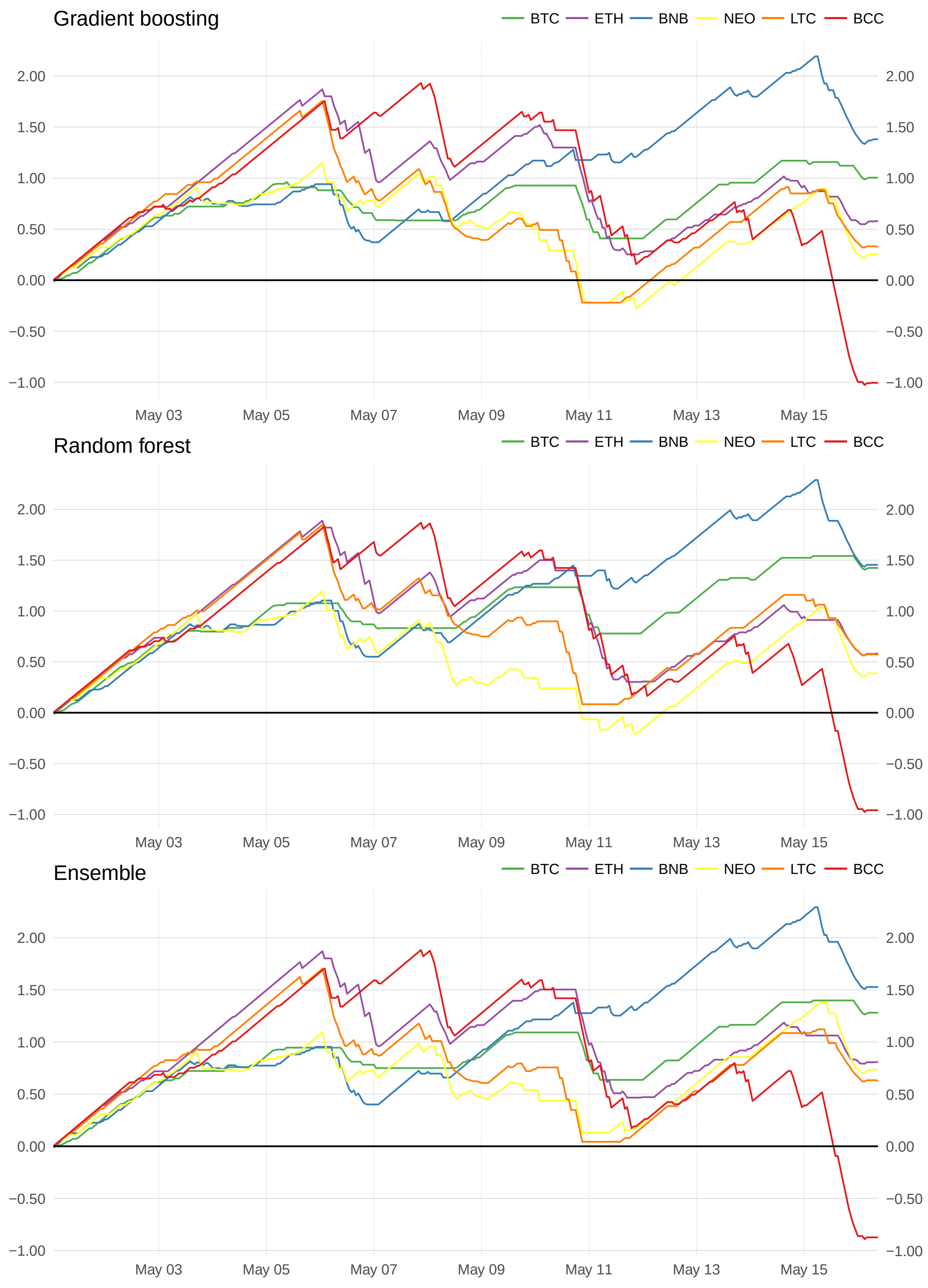


Figure 8.7: The cumulative returns from trading based on GB, RF, and Ens across the BTC, ETH, BNB, NEO, LTC, and BCC pairs in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59.

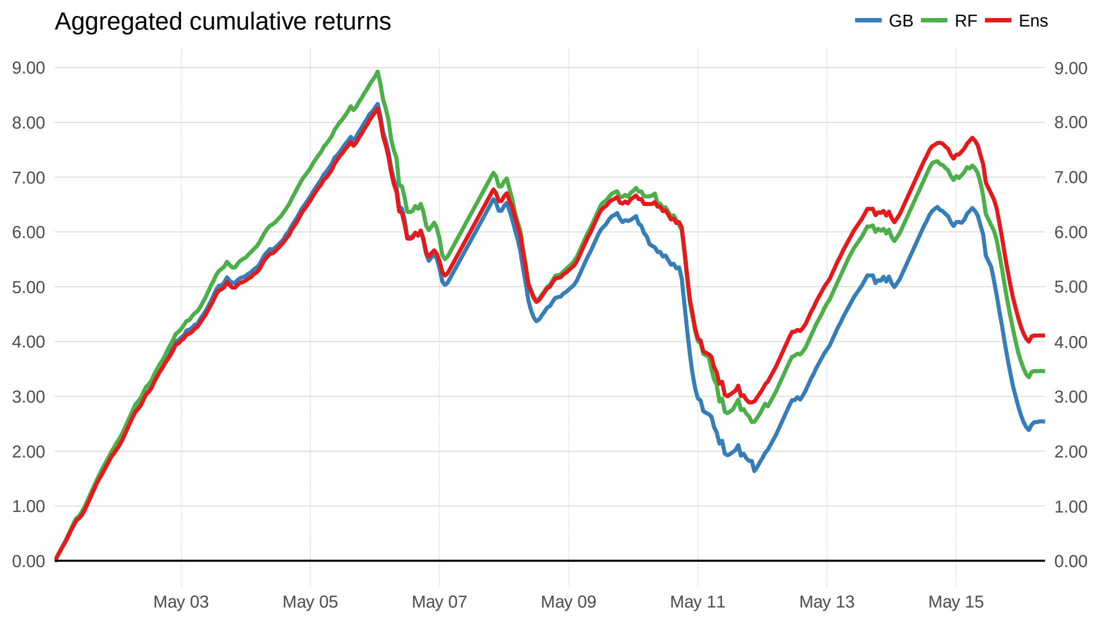


Figure 8.8: The cumulative returns from trading based on GB, RF, and Ens aggregated over the BTC, ETH, BNB, NEO, LTC, and BCC pairs in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59.

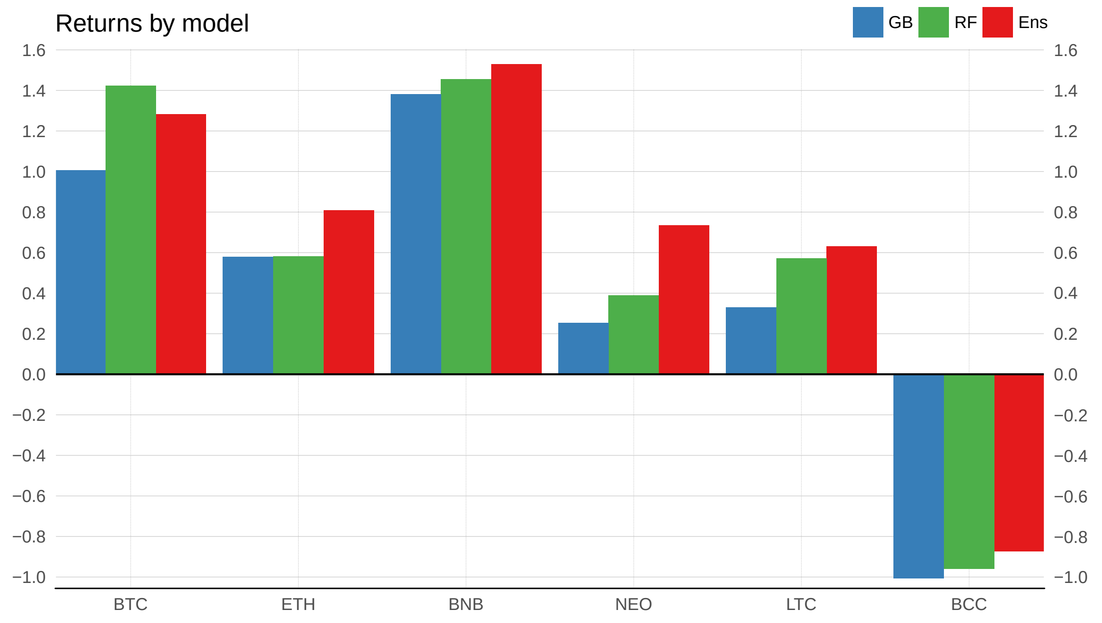


Figure 8.9: Barplots depicting the returns from trading based on GB, RF, and Ens across the BTC, ETH, BNB, NEO, LTC, and BCC pairs in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59.

	GB	RF	Ens
Buys	162	168	149
True buys	109	117	107
False buys	53	51	42
Stays	207	201	220
True stays	158	160	169
False stays	49	41	51
Losses	29	23	19
Accuracy	0.72	0.75	0.75
Fees	0.33	0.34	0.30
Return	1.00	1.42	1.28

Table 8.4: BTC

	GB	RF	Ens
Buys	288	287	274
True buys	215	217	211
False buys	73	70	63
Stays	81	82	95
True stays	49	52	59
False stays	32	30	36
Losses	61	60	53
Accuracy	0.72	0.73	0.73
Fees	0.58	0.58	0.55
Return	0.58	0.58	0.81

Table 8.5: ETH

	GB	RF	Ens
Buys	268	283	257
True buys	195	204	192
False buys	73	79	65
Stays	101	86	112
True stays	60	54	68
False stays	41	32	44
Losses	56	59	50
Accuracy	0.69	0.70	0.70
Fees	0.54	0.57	0.52
Return	1.38	1.45	1.53

Table 8.6: BNB

	GB	RF	Ens
Buys	260	268	247
True buys	194	201	190
False buys	66	67	57
Stays	109	101	122
True stays	76	75	85
False stays	33	26	37
Losses	60	62	52
Accuracy	0.73	0.75	0.75
Fees	0.52	0.54	0.50
Return	0.25	0.39	0.73

Table 8.7: NEO

	GB	RF	Ens
Buys	286	285	268
True buys	207	209	198
False buys	79	76	70
Stays	83	84	101
True stays	51	54	60
False stays	32	30	41
Losses	70	67	62
Accuracy	0.70	0.71	0.70
Fees	0.57	0.57	0.54
Return	0.33	0.57	0.63

Table 8.8: LTC

	GB	RF	Ens
Buys	318	325	307
True buys	248	256	242
False buys	70	69	65
Stays	51	44	62
True stays	26	27	31
False stays	25	17	31
Losses	67	67	63
Accuracy	0.74	0.77	0.74
Fees	0.64	0.65	0.61
Return	-1.01	-0.96	-0.87

Table 8.9: BCC

Table 8.10: Trade summaries from trading based on GB, RF, and Ens across the BTC, ETH, BNB, NEO, LTC, and BCC pairs in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59.

9 | Concluding Remarks

In this thesis, what we have essentially done is set up a framework that allows easy model development and backtesting for trading cryptocurrencies, and shown its potential effectiveness on multiple trading pairs. At the heart of the framework is the hypothesis that some time dependent function exists, that given the right information can predict future profits on cryptocurrencies. We find evidence supporting the existence of such a function by producing profits on five of six cryptocurrency pairs, with models derived using the BTC-USDT pair. We show that 1 minute trading data aggregated into 1 hour observations can serve as proxy for the information needed for the predictive function. We further show that in this particular setup gradient boosting and random forests outperform to GLM and neural networks.

Through the model derivation we show the local nature of the predictive function by increasing profits through a reduction in the training set size. For gradient boosting the evidence supporting a reduction of the training set was not unanimous since there was a decrease in profits, however, we assumed this was a data related coincidence. To further support this assumption we applied the gradient boosting model without the reduction of training set size to the new data for all six pairs, which resulted in losses across all of them.

Staying true to the spirit of both gradient boosting and random forests we created an ensemble model by combining the two models, which overall outperforms both gradient boosting and random forests individually.

In Section 9.1 we discuss the return on investment from actually implementing the trading framework and in Section 9.2 we briefly touch on extending the framework.

9.1 | Estimating Cost and Return

The profits reported throughout the thesis are rather large percentages, however, for actual implementation it is important to keep in mind that profits are calculated as a percentage of the fixed amount of each trade. In the following two sections we calculate the return on investment by first estimating the required liquidity for initializing, which we subsequently use to estimate return on investment.

9.1.1 | Initialization Cost

Throughout the result presentations we do not touch on the subject of how much liquidity is needed to actually make the reported profits, we simply assume the trading account used already has enough liquidity to place the required orders. Here we estimate the liquidity required to apply the framework, based on how many trading pairs are included and the trade horizon. Each time a trade is performed there must be sufficient funds to buy a specific amount of the asset, and at same time already have that same amount of the asset to set the stop-limit order. Once a trade has been made, the liquidity used for that trade can be locked down in a time period of which the upper limit is defined by the trade horizon chosen. The estimated initialization cost is then

$$\text{pairs} \cdot \text{amount} \cdot (\text{horizon} + 1) \cdot 2 = \text{initialization cost},$$

where we add one to the horizon to provide some wiggle room in case the first couple of trades are losses. For the parameters used in this thesis, i.e., six trading pairs and a 24 period trade horizon, the initialization cost is

$$6 \cdot \text{amount} \cdot (24 + 1) \cdot 2 = 300 \cdot \text{amount},$$

where the amount per trade could be fixed in terms of USDT value, or in terms of the cryptocurrency used for trading.

9.1.2 | Return on Investment

Let us now assume that at each trade we buy an amount of the given asset corresponding to 100 USDT. This assumption implies that we need to make an initial investment of

$$300 \cdot 100 = 30000 \text{ USDT}.$$

Assume that we use the ensemble model which yields a profit of 400% of the average investment, a profit of 400 USDT, over the course of roughly two weeks. That is a monthly return of approximately 2.6%, which is not a bad return and could most likely be improved by deriving models specific to each trading pair. Since we are required to have half the initial investment placed in the cryptocurrencies we trade, we are exposed to movements in the USDT value of these cryptocurrencies. As such, we recommend only using this framework on trading pairs you see increase in the long run (or at least until you want to cash out). Alternatively, the downside protection provided by stop-limits could be removed, which would half the initialization cost and remove the exposure from holding the traded cryptocurrencies. In Section 9.2.3 we present a third option that could potentially circumvent the additional initialization costs and risk exposure from stop-limits while still providing some downside protection.

9.2 | Topics for Further Development

In this section we present some of the ideas we have for further developing the framework and models.

9.2.1 | Local Data Parametrization

We find evidence to support the local market dynamics hypothesis, i.e., some local explanatory variables are able to predict when to buy and stay. As such, it would be wise to examine the data parametrization periodically to ensure the optimal combination is used. This would consist of using the presented framework to further examine which limit, stop-limit, horizon, etc. is best suited for a given period for a given trading pair and then perform a finer search around these values. We consider five limits, for example, and find that a 2% limit performs best, we could then examine a finer interval around 2% perhaps improving the true classification rate, and likewise for the other parameters. We found that a 2% limit and 10% stop-limit seemed a decent combination but our search was not exhaustive, thus, this combination is unlikely the ideal. The ideal combination is also likely to change over time and across trading pairs.

Another way we think might increase the percentage of true buys is to trade using a lower limit than is used for classification. An example would be to classify and predict buys with a limit of 3% but then set the actual limit orders at 2%, the motivation being that we are predicting a higher increase than we are aiming for, which could perhaps lead to more limit orders being triggered.

We also imagine that combining multiple aggregation intervals could further improve the true classification rate. This would consist of setting up separate models, customized to each aggregation interval considered and only trade when the models agree on trading. Consider the 1 hour aggregation interval, which would give a new prediction every hour, and combine this with the 15 minute interval, which would give four predictions per hour. One way to combine the two intervals would be to only allow the 15 minute model to trade on candles which the 1 hour model classifies as buys.

9.2.2 | Local Model Configuration

Similar to the local data parametrization described in Section 9.2.1, a periodical model recalibration to determine the optimal model calibration should be used. The framework allows easy backtesting to reconfigure the models and examine which were more profitable in a given period. Assuming the local market dynamics hypothesis is true it seems likely that the model configuration is also subject to change over time.

The results presented in this thesis are all based on models derived using only the BTC-USDT pair. It is likely that the results would improve had we used pair-specific model configurations for each pair.

9.2.3 | Reversing the Framework

In the presented framework we only consider the prediction of price increases. We use the implementation of stop-limit orders to reduce potential downside risk, which as mentioned in Section 9.1 has the consequence that the trading account must have a sufficient holding of a given asset to place these. While further exposing the trader to changes in the USDT value of the cryptocurrencies that must be held.

A possible way to obtain downside protection, without the additional initialization cost and exposure, is to flip the framework and instead of predicting when a price is likely to increase, predict when it is likely to decrease. By predicting decreases we could potentially avoid using stop-limit orders and instead simply liquidate the trades whenever the model predicts eminent price decreases.

9.2.4 | Technical Analysis Models

For the TA factors added to the trading data we only considered parameters that were typically used for the moving averages, and only one way to deduce the trading signals from the factors. We do find some evidence that the trading factors can assist the models to further improve profits, and as such, it might be worth considering further parametrization and trading rules for the TA factors. A TA model could even be constructed by combining the different trading signals of the three TA indicators and more could be added to further optimize this model.

9.2.5 | Crypto-to-Crypto Trading

While we do perform crypto-to-crypto trading in this thesis, we trade against USDT, a rather stable cryptocurrency. Assume that we want to increase our holdings of BTC and ETH. We could then initially split our holdings into half BTC and half ETH. Say we also managed to construct a model to predict decreases in the USDT value of either of the two. Then if the models predict that the USDT value of ETH would increase and the USDT value of BTC decrease, we could move our holdings into ETH. Conversely, when we expect the BTC to increase and ETH decrease we could move our holdings back into BTC. Using this setup would half the fees that we would have to pay if we were simply trading BTC-USDT and ETH-USDT simultaneously. Trading crypto-to-crypto also opens up for the trading of other pairs that do not trade against USDT.

Appendices

A | Technical Analysis Factors

We base the TA factors on the *exponential moving average* (EMA) to give more weight to more recent observations than the *simple moving average* (SMA). Throughout this section we denote closing price by p_t and the EMA is then defined as

$$EMA_t(p_t, n) = \begin{cases} p_t, & t = 1 \\ K \cdot p_t + (1 - K) \cdot EMA_{t-1}, & t > 1. \end{cases}$$

where

$$K = \frac{2}{n+1}$$

From the EMA definition it is clear that it can be estimated without dropping observations, however, in our implementation we initialize the EMA by inserting a 14 period SMA instead of p_t in the case where $t = 1$. We cover the derivation of TA signals in the following sections, all TA signal are calculated using closing prices. The implementation of deriving the TA factor and adding them to data are shown in Appendix B.2.4. The parameters chosen when calculating any of the trade signals are all subjective, the implementation below depicts what in our experience are generally popular choices. The relevant theory for each indicator is based on the implentation used in the TTR R-package by Ulrich (2017).

A.1 | Relative Strength Index

The RSI is a popular momemtum oscillator which moves within a range from 0 to 100 and has two noteworthy zones. When the RSI is above 70 the underlying asset is considered overbought, and it is likely that the price will decline. When the RSI is below 30 the underlying asset is considered oversold and it is likely that the price will start increasing. It is trypically recommended to use a 14 period RSI, which is what we consider here, however increasing this period will make the RSI less sensitive to changes, and decreasing it, more sensitive.

The RSI is calculated by measuring the average gains and losses over the specified period and creating a ratio from these that is then charted to provide the RSI plot in Figure A.1. The RSI is calculated as

$$RSI_t = 100 - \frac{100}{1 + RS_t},$$

where RS is the relative strength measured by an EMA of average gains (AG) over an EMA of average losses (AL) given by

$$RS_t = \frac{EMA_t(AG_t, 14)}{EMA_t(AL_t, 14)}.$$

The AG and AL are defined as

$$AG_t = \max(0, p_t - p_{t-1}),$$
$$AL_t = \min(0, p_t - p_{t-1}).$$

Using the RSI, typical bullish signals are when the RSI crosses from below 30 to above,

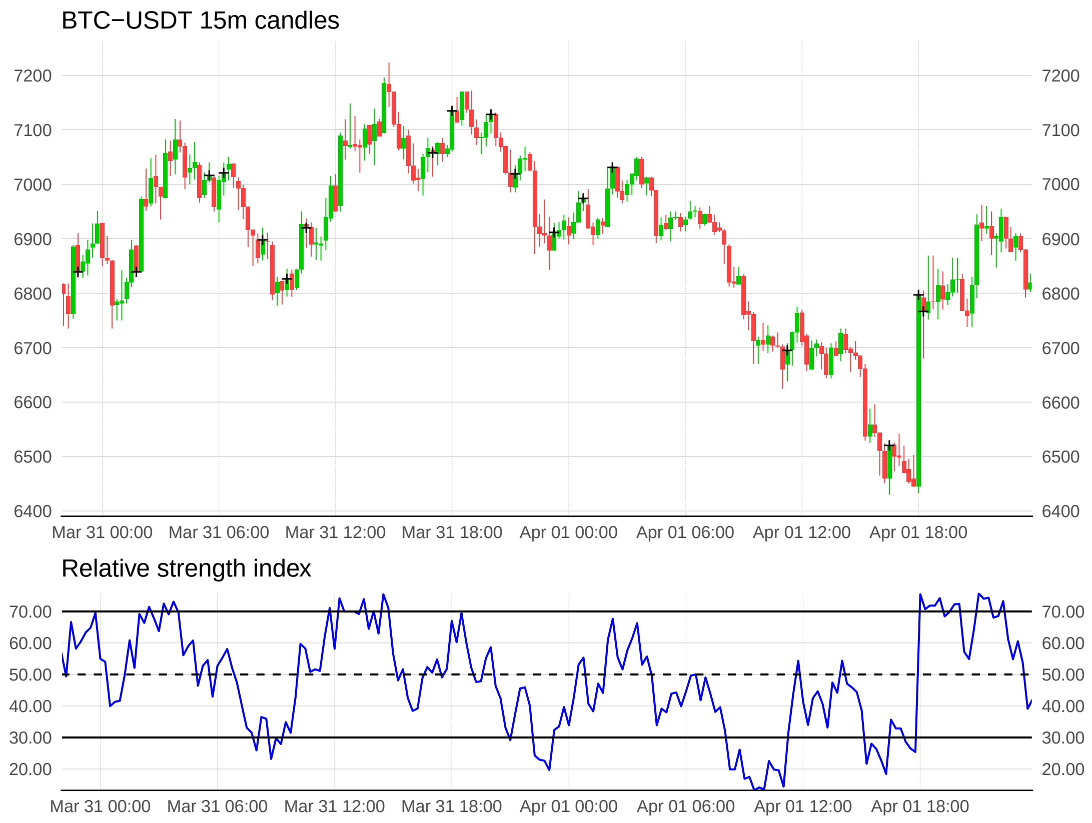


Figure A.1: BTC-USDT 15m candles in the period from March 30th, 2018 at 22:00 to April 1st, 2018 at 23:45 and the RSI of the same period. The candles we buy on according to the RSI are marked with a "+".

and when it crosses from below 50 to above. To derive the trading signals from the RSI we construct two sets of conditions, which if true are considered buy signals. The first case is when

$$RSI_t \geq 30,$$
$$RSI_{t-1} < 30,$$

and the second case is when

$$RSI_t \geq 50,$$
$$RSI_{t-1} \geq 50,$$
$$RSI_{t-2} < 50.$$

A.2 | Moving Average Convergence / Divergence

The MACD consists of two lines, the MACD and the signal line, both based on the closing prices of each candle. The MACD line is calculated by subtracting a longer EMA from a shorter EMA, typical periods used for these moving averages are 26 for the longer and 12 for the shorter, thus, the MACD is calculated as

$$MACD_t = EMA_t(p_t, 12) - EMA_t(p_t, 26).$$

The signal line is typically calculated as a 9-period EMA of the MACD and charted ontop of the MACD line

$$Signal = EMA_t(MACD_t, 9).$$

Using the MACD, typical bullish signals are when the MACD line crosses above the

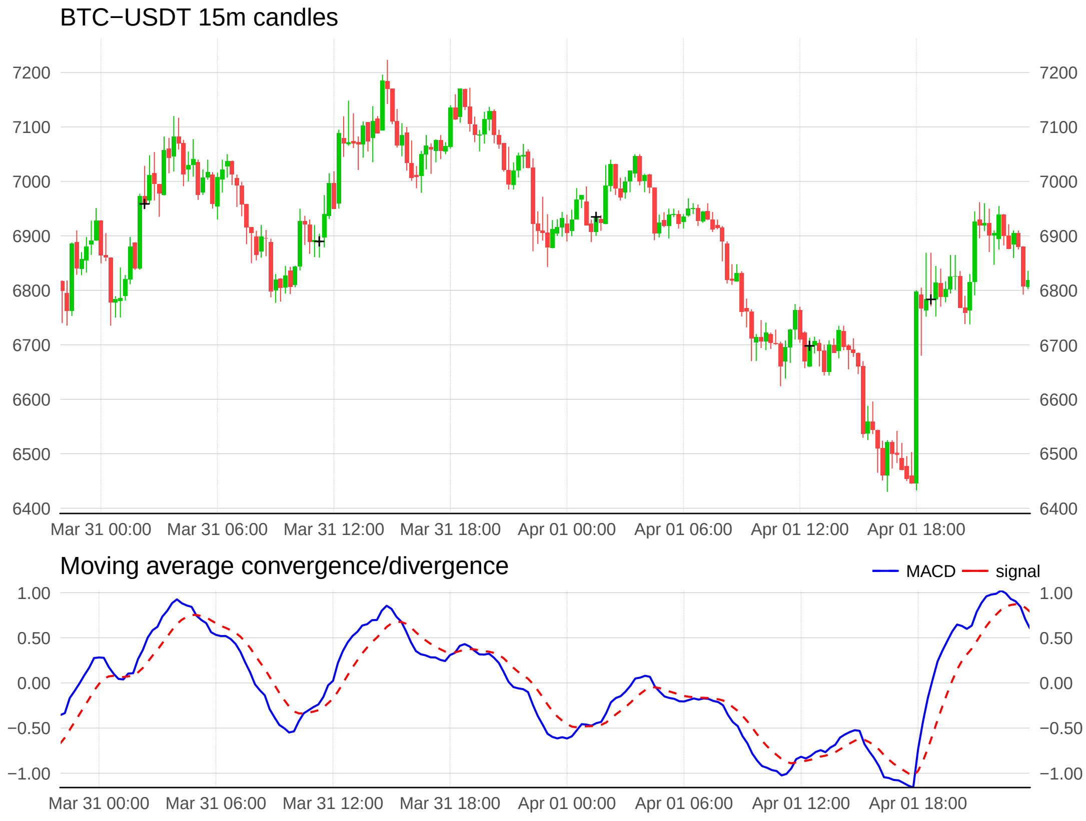


Figure A.2: BTC-USDT 15m candles in the period from March 30th, 2018 at 22:00 to April 1st, 2018 at 23:45 and the MACD of the same period. The candles we buy on according to the MACD are marked with a "+".

signal line. Often investors wait for a few periods to confirm the cross is true before entering a long position, as such we consider a 3-period filter, meaning we wait for the MACD line to have been above the signal line for at least 3 periods after crossing.

According to the MACD we buy when the following conditions are met

$$\begin{aligned} MACD_t &> Signal_t, \\ MACD_{t-1} &> Signal_{t-1}, \\ MACD_{t-2} &> Signal_{t-2}, \\ MACD_{t-3} &\geq Signal_{t-3}, \\ MACD_{t-4} &< Signal_{t-4}. \end{aligned}$$

Using these calculations, both the MACD and Signal line are charted to provide the signal shown in Figure A.2.

A.3 | Average Directional Index

The ADX is used to determine the strength of the price movement trend and takes on values between 0 and 100 indicating weak and strong trends, though it rarely exceeds 60. Values between 0 and 25 indicate an absent or weak trend, and values above 25 indicate an increasingly stronger trend. The ADX is a non-directional trend indicator, and is often charted in conjunction with the two directional indexes (DIs) from which the trade signal is derived. To calculate the ADX we start by calculating two directional movement indexes (DMIs), DMI^+ and DMI^-, which are based on the price changes for each candle. The DMIs are calculated by considering the one-period changes to the highest price, h_t, and lowest price, l_t, of the candles

$$\begin{aligned} \Delta High_t &= h_{t-1} - h_t, \\ \Delta Low_t &= l_t - l_{t-1}. \end{aligned}$$

The directional movements are then calculated using the following three cases. If $\Delta High_t < 0$ and $\Delta Low_t < 0$, or $\Delta High_t = \Delta Low_t$ then

$$\begin{aligned} DMI_t^+ &= 0, \\ DMI_t^- &= 0. \end{aligned}$$

If $\Delta High_t > \Delta Low_t$ then

$$\begin{aligned} DMI^+ &= \Delta High_t, \\ DMI^- &= 0. \end{aligned}$$

Finally if $\Delta High_t < \Delta Low_t$ then

$$\begin{aligned} DMI^+ &= 0, \\ DMI^- &= \Delta Low_t. \end{aligned}$$

A true range, TR, is then calculated as the true high, TH, minus the true low, TL, that is

$$TR_t = TH_t - TL_t,$$

where

$$\begin{aligned} TH_t &= \max(h_t, p_{t-1}), \\ TL_t &= \min(l_t, p_{t-1}), \end{aligned}$$

where p_t still denoted closing price. A Wilder Welles EMA (WEMA) is then applied to DMI^+, DMI^-, and TR, which is simply an EMA with weighting coefficient $K = \frac{1}{n}$ instead of the usual $K = \frac{2}{n+1}$, and two directional indicators, DI^+ and DI^-, are derived. Typically a 14-period WEMA is used, making the directional indicators

$$\begin{aligned} DI_t^+ &= 100 \times \frac{WEMA_t(DMI_t^+, 14)}{WEMA_t(TR_t, 14)}, \\ DI_t^- &= 100 \times \frac{WEMA_t(DMI_t^-, 14)}{WEMA_t(TR_t, 14)}. \end{aligned}$$

The directional movement index (DX) is then calculated as

$$DX_t = \frac{DI_t^+ - DI_t^-}{DI_t^+ + DI_t^-},$$

and a 14-period EMA is applied to arrive at the ADX. Finally the ADX, DI^+, and DI^- are charted to give the visual shown in Figure A.3. Using the ADX, typical bullish signals are when the DI^+ crosses above the DI^-, while the ADX is above 25, suggesting a strong trend. In order to try and eliminate buying when the trend is strongly diminishing, we consider only the cases where the ADX has increased for in one of three precious periods. According to the ADX we buy when the two conditions

$$\begin{aligned} DI_t^+ &> DI_t^-, \\ ADX_t &> 25, \end{aligned}$$

and one of the following are met

$$\begin{aligned} ADX_t &> ADX_{t-1}, \\ ADX_{t-1} &> ADX_{t-2}, \\ ADX_{t-2} &> ADX_{t-3}. \end{aligned}$$

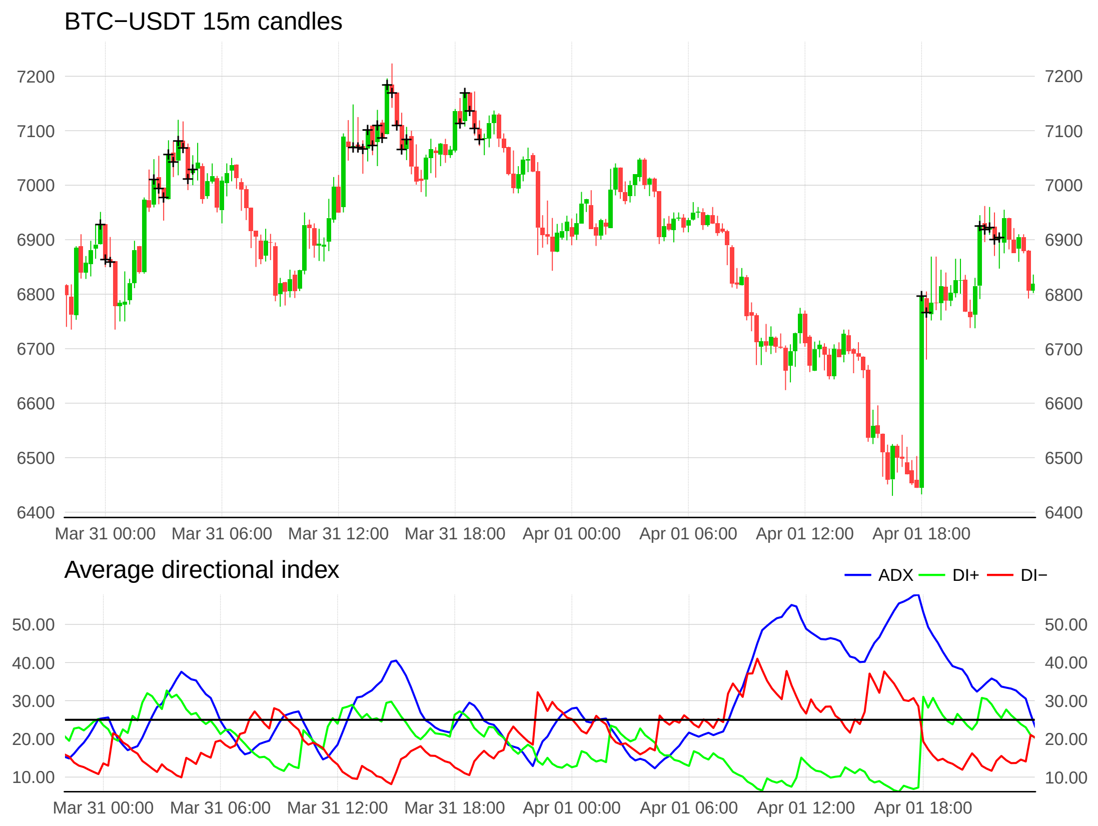


Figure A.3: BTC-USDT candles in the period from March 30th, 2018 at 22:00 to April 1st, 2018 at 23:45 and the ADX, DI^+, and DI^- of the same period. The candles we buy on according to the ADX are marked with a '+'.

B | Code

B.1 | R-Packages

Below we list all R-packages used in the thesis, split into areas of applicability.

```r
# Wrangling and Computing
library(dplyr)              # Data Manipulation
library(magrittr)           # Pipe-Operators
library(tidyr)              # Data Tidying
library(reshape2)           # Data Reshaping
library(tibble)             # Data Frame Format
library(readr)              # Read Data
library(foreach)            # Iterative Computing
library(doParallel)         # Parallel Backend for foreach
library(compiler)           # Byte Code Compiler
library(purrr)              # Functional Programming

# API and Database
library(httr)               # HTTP Requests
library(digest)             # Hash functions

# Modelling
library(glmnet)             # GLM and Penalized Regression
library(randomForest)       # Random Forest
library(xgboost)            # Gradient Boosting
library(keras)              # Neural Networks
library(TTR)                # Technical Analysis
library(pROC)               # ROC-Curve

# Plotting and tables
library(ggplot2)            # Plotting Environment
library(grid)               # Plot Grid
library(gridExtra)          # Grid Arranging
library(gtable)             # Grid Alignment
library(corrplot)           # Correlation plot
library(xtable)             # Exporting LaTeX Tables

# Date and Time
library(anytime)            # Date and Time Conversion
library(lubridate)          # Date and Time Calculations

# Misc
library(Hmisc)              # Capitalize String
```

B.2 | Framework Implementation

In this appendix we present the implementation of the code neccessary for using the data parametrization framework described in Chapter 1, as well as a short description of the arguments used in each function.

B.2.1 | The Binance API

To obtain trading data from the Binance API kline endpoint, we first set up a function that makes a single GET request to the API. It takes the trading pair `symbol`, aggregation `interval`, and `startTime` arguments.

```r
# Binance candlesticks Timed
Binance_Candlesticks_Timed <- function(
  symbol, interval, startTime){

  # Set URL to API endpoint
  api_url <- "https://api.binance.com"
  req_url <- "api/v1/klines"

  # Define the parameters for the API call
  params <- list(symbol = symbol,
                 interval = interval,
                 startTime = startTime)

  # Make the call
  response <- content(GET(api_url, path = req_url, query = params))

  # Restructure the response
  response_df <- as.data.frame(
    foreach(i = 1:length(response), .combine = rbind) %do% {
      foreach(j = 1:12, .combine = c) %do% {
        response[[i]][j]
      }
    }
    , stringsAsFactors = FALSE, row.names = FALSE)

  # Filter and name the response
  response_df <- response_df[,-12]
  cols_numeric <- c(1,2,3,4,5,6, 7 ,8,9,10,11)
  response_df[, cols_numeric] = apply(response_df[, cols_numeric], 2, function(x) as.numeric(x))

  colnames(response_df) <- c("Open_time",
                             "Open",
                             "High",
                             "Low",
                             "Close",
                             "Volume",
                             "Close_time",
                             "Quote_asset_volume",
                             "Number_of_trades",
                             "Taker_buy_base_asset_volume",
                             "Taker_buy_quote_asset_volume")

  # Return the response
  return(response_df)
}
```

We then nest this GET reqeust function in another function, which makes as many GET requests as needed to get all data available between the startTime and endTime. This function takes the trading pair`symbol`, aggregation `interval`, and `startTime`, and `endTime` arguments.

```r
# Binance Candlesticks Historical
Binance_Candlesticks_Historical <- function(
  symbol, interval, startTime, endTime){

  # Check inputs
  if(endTime - startTime <= 0) stop("startTime must be before endTime!")

  # Initial data setup
  data <- Binance_Candlesticks_Timed(symbol = symbol, interval = interval, startTime = startTime)

  # Set start time for while loop
  next_startTime <- startTime + 60000000

  # Get all full api calls
  while(next_startTime <= endTime){
    next_data <- Binance_Candlesticks_Timed(symbol = symbol, interval = interval,
                                            startTime = next_startTime)
    data <- rbind(data, next_data)
    next_startTime <- next_startTime + 60000000
  }

  # Cut data to start and end time
  data <- data[which(data$Open_time <= endTime),]

  return(data)
}
```

Finally, we set up a top-level function that downloads all data for multiple trading pairs using the GET reqeust function, and saves the data in a specified path. This function takes a vector of trading pair `symbols`, aggregation `interval`, and `startTime`, `endTime`, and `path_save` to save the data as inputs

```r
# Get Candlesticks
Get_Candlesticks <- function(
  pairs, interval, startTime, endTime, path_save = NULL){

  # Download all trading pairs for the specified period
  start_timer <- Sys.time()
  data <- foreach(i = pairs, .packages = c("httr", "foreach"),
                  .export = c("Binance_Candlesticks_Historical", "Binance_Candlesticks_Timed")) %dopar% {
                    response <- Binance_Candlesticks_Historical(symbol = i, interval = interval,
                                                                startTime = startTime, endTime = endTime)
                    if(nrow(response) == 0){
                      stop(paste0("Trading pair ", i, " has no data for this period!"))
                    }
                    cat(paste0("Trading pair ", i, " downloaded."))
                    return(response)
                  }
  names(data) <- pairs
  end_timer <- Sys.time()
  time_taken <- end_timer - start_timer
  cat(paste0("Downloaded ", length(pairs), " trading pairs in"), time_taken, "\n")

  # If no path for saving is provided, download data to R object
  if(is.null(path_save)){
    cat(paste0("All ", length(pairs), " pairs downloaded as R object."), "\n")
    return(data)
  }
  # If path for saving is provided, save data to path
  else {
    foreach(i = pairs) %dopar% {
      # Download data
      temp_data <- data[[i]]

      # Save the candlesticks for current the trading pair
      write.csv(
        x = temp_data,
        file = file.path(path_save, paste0(i, ".csv")),
        row.names = FALSE, quote = TRUE
      )
      cat(paste0("Trading pair ", i, " saved as .csv file."))
    }
    cat(paste0("All ", length(pairs), " pairs saved to ", path_save), "\n")
    return(time_taken)
  }
}
```

B.2.2 | Data Preparation

To prepare the raw data downloaded using the code in Appendix B.2.1 we use the top-level function presented below. It parametrizes the raw data in all possible combinations of the inputs given, as described in Section 1.4, and uses the functions presented in Appendices B.2.3-B.2.7. This function also handles the problem of making the datasets the same size regardless of difference orders, lag orders, and whether or not factors are added, as well as the profit scouting described in Section 7.1. It takes 18 arguments, the first two of which are unique to the function, `filter` is the threshold for filtering the datasets by potential profit and `candlesticks` is the raw data. The remaining arguments correspond to the arguments used by functions in Appendices B.2.3-B.2.6.

```r
Prepare_Candlesticks <- function(filter = 1.00, candlesticks,
  pairs = NULL, interval = "1m", only_full = FALSE, # Aggregate_Candlesticks
  factors_based_on = "Close", factors = FALSE, time_factor = FALSE, exclude_na = TRUE, # AddTo_Candlesticks
  classify_based_on = "Close", limit = 0.01, stop = 0.02, horizon = 0, # Classify_Candlesticks
  diff_value = 0, max_diff = 0, lag = 0, n_test = 0.2, exclude = FALSE){ # Split_Candlesticks
  # Check filter input
  if(filter > 1.00) stop("Filter value must be between 0.00 and 1.00")

  # Create multiple datasets for each trading pair
  pairs <- names(candlesticks)

  # Set all parameter combinations
  max_lag = max(lag)
  parameters <- expand.grid(
    # Aggregate_Candlesticks
    interval = interval,
    # AddTo_Candlesticks
    time_factor = time_factor,
    diff_value = diff_value,
    factors = factors,
    # Classify_Candlesticks
    limit = limit,
    stop = stop,
    # Split_candlesticks
    lag = lag,
    exclude = exclude,
    stringsAsFactors = FALSE)

  # Set ID for each parameter combinations and add ID column
  n_parameters <- nrow(parameters)
  parameters <- cbind(ID = c(1:n_parameters),
                      Buys = rep(NA, n_parameters),
                      Stays = rep(NA, n_parameters),
                      Profits = rep(NA, n_parameters),
                      parameters)

  # Set amount of sets to return after filtering and check if filter would return any sets at all
  n_filter <- round(n_parameters * filter, 0)
  if(n_filter < 1){
    n_filter <- 1
    cat("Filter value of", filter, "would reuturn 0 sets. Returning the most profitable instead. \n")
  }
  cat("Starting parametrization for", n_parameters, "different combinations for each trading pair.\n")

  # Prepare candlesticks for each pair
  results <- foreach(pair = pairs) %do% {
    cat(pair, "start \n")
    # Aggregate datasets only once
    aggregated_dfs <- foreach(unqiue_interval = interval) %do% {
      Aggregate_Candlesticks(
        candlesticks = candlesticks,
        pairs = pair,
        interval = unqiue_interval,
        only_full = only_full)
    }
    names(aggregated_dfs) <- interval
    cat("All aggregation levels complete \n")

    # Prepare dataset for each parameter combination
    parameter_sets <- foreach(
      parameter_row = c(1:n_parameters),
      .export = c("AddTo_Candlesticks", "Classify_Candlesticks", "Split_Candlesticks", "Calculate_Profit"),
      .packages = c("foreach")) %dopar% {
        # Set current parameters to be used
        current_parameters <- parameters[parameter_row, ]


        # Check if factors are added, if they are also handle timestamp and direction
        if(current_parameters$factors == FALSE){
          current_exclude <- TRUE
          current_time_factor <- FALSE
        } else {
          current_exclude <- FALSE
          current_time_factor <- TRUE
        }
        if(current_exclude == FALSE) current_parameters$exclude <- current_exclude
        current_parameters$time_factor <- current_time_factor

        # Add factors to candlesticks
        factors_df <- AddTo_Candlesticks(
          candlesticks = aggregated_dfs[[current_parameters$interval]],
          based_on = factors_based_on,
          factors = current_parameters$factors,
          time_factor = current_parameters$time_factor,
          exclude_na = exclude_na)

        # Classify candlesticks
        classified_df <- Classify_Candlesticks(
          candlesticks = factors_df,
          based_on = classify_based_on,
          limit = current_parameters$limit,
          stop = current_parameters$stop,
          horizon = horizon)

        # Account for factors
        classified_df[[pair]] <-
          classified_df[[pair]][ifelse(current_parameters$factors, 4, 37):nrow(classified_df[[pair]]),]

        # Split candlesticks
        split_df <- Split_Candlesticks(
          candlesticks = classified_df,
          diff_value = current_parameters$diff_value,
          max_diff = max_diff,
          lag = current_parameters$lag,
          n_test = n_test,
          exclude = current_exclude,
          factors = current_parameters$factors)

        # Get prepared data to return
        prepared_data <- split_df[[pair]]

        # Calculate potential profits
        calculated_profits <- Calculate_Profit(data = split_df[[pair]], set = "scouting",
                                               parameters = current_parameters, horizon = horizon,
                                               ignore_stops = FALSE, PL = FALSE, fee = 0.001)
        current_parameters$Buys <- calculated_profits$n_buys
        current_parameters$Stays <- calculated_profits$n_stays
        current_parameters$Profits <- calculated_profits$profit

        result <- list(prepared_data, current_parameters)
        names(result) <- c("Data", "Parameters")
        return(result)
      }
    names(parameter_sets) <- as.character(parameters$ID)
    cat("All computations complete \n")

    # Collect parameters from each dataset and sort parameters by potential profit in training set
    pair_parameters <- foreach(row_parameter = c(1:n_parameters), .combine = rbind) %do% {
      parameter_sets[[row_parameter]]$Parameters
    } %>% arrange(desc(Profits), stop)

    # Filter the parametrized sets by profit
    if(filter < 1.00){
      # Set threshold for profits to ensure equally profitable pairs are not excluded
      profit_threshold <- pair_parameters[n_filter, "Profits"]

      # Filter parametrized sets by profit
      pair_parameters <- pair_parameters[which(pair_parameters$Profits >= profit_threshold),]
      parameter_sets <- parameter_sets[pair_parameters$ID]

      if(filter == 0.0){
        pair_parameters <- pair_parameters[which(pair_parameters$stop == min(pair_parameters$stop)),]
        parameter_sets <- parameter_sets[pair_parameters$ID]
      }
    }
    cat("All filtering complete \n")
    data <- list(parameter_sets, pair_parameters)
    names(data) <- c("Sets", "Parameters")
    cat(pair, "end \n")
    return(data)
  }
  names(results) <- pairs
  return(results)
}
```

B.2.3 | Aggregation

To aggregate the 1m candles obtained using the code in Appendix B.2.1 we use the function below. It takes four arguments, `candlesticks` is the raw 1m data, `pairs` can be used to only aggregate select trading pairs, if `pairs` is NULL all trading pairs will be aggregated, aggregation `interval` is the desired interval to aggregate, and `only_full` controls whether or not to exclude candles with less than full information.

```r
Aggregate_Candlesticks <- function(
  candlesticks, pairs = NULL, interval = "1m", only_full = FALSE){

  # Set intervals, corresponding minutes, and globals
  intervals <- c("1m", "5m", "15m", "30m", "1h", "2h", "4h", "8h", "12h", "24h")
  minutes <- c(1, 5, 15, 30, 60, 120, 240, 480, 720, 1440)
  if(!(interval %in% intervals)){
    stop("Interval not implemented - Try 1m, 5m, 15m, 30m, 1h, 2h, 4h, 8h, 12h, 24h.")
  }
  aggregate <- minutes[match(interval, intervals)]
  if(is.null(pairs)) pairs <-  names(candlesticks)

  # Aggregate candlesticks
  data <- foreach(i = pairs, .packages = c("foreach", "anytime")) %do% {
    # Get specific pair and check if aggregation is possible
    raw_df <- candlesticks[[i]]

    # Check the data and if aggregation interval is possible
    check_row <- nrow(raw_df)
    check_full_candle <- floor(check_row/aggregate)
    if(check_full_candle == 0){
      stop(paste0("Cannot aggregate ", interval, " candle with only ", check_row, " minutes of data."))
    }
    if(check_row < 61) stop("Dataset must contain more than 60 minutes of data.")

    # Get desired variables from raw data
    temp_df <- raw_df[, c(1:6, 9)]
    colnames <- c("Time", "Open", "High", "Low", "Close", "Volume", "Trades")
    colnames(temp_df) <- colnames
    temp_df$Time <- anytime(temp_df$Time/1000)

    # Aggregate candles if needed
    if(aggregate == 1){
      aggregated_candles <- temp_df
    } else {
      # Determine how many candles to make and handle sparse first and last candles
      min_prior <- ifelse(aggregate <= 60,
                          min(which(diff(lubridate::hour(temp_df$Time)) == 1)),
                          min(which(diff(lubridate::day(temp_df$Time + hours(1))) == 1)))
      candle_first_size <- min_prior %% aggregate
      candles_middle_from <- candle_first_size + 1
      candles_middle_amount <- floor((check_row - candle_first_size)/aggregate)
      candle_last_size <- (check_row - candle_first_size) %% aggregate

      # Aggregate candles
      aggregated_candles <- foreach(j = 1:candles_middle_amount, .combine = rbind) %do% {
        # Set candles to be aggregated
        candle_from <- candles_middle_from + (j * aggregate - aggregate)
        candle_to <- candle_from + aggregate - 1
        candle_data <- temp_df[candle_from:candle_to,]

        # Aggregate candles to desired interval
        candle_middle <- data.frame(
          Time <- candle_data$Time[1],
          Open <- candle_data$Open[1],
          High <- max(candle_data$High),
          Low <- min(candle_data$Low),
          Close <- candle_data$Close[aggregate],
          Volume <- sum(candle_data$Volume),
          Trades <- sum(candle_data$Trades),
          stringsAsFactors = FALSE)
        colnames(candle_middle) <- colnames

        return(candle_middle)
      }

      if(!only_full){
        # If sparse first candle, calculate it
        if(candle_first_size != 0){
          # Set candles to be aggregated
          candle_data <- temp_df[1:candle_first_size,]
          # Aggregate first sparse candle
          candle_first <- data.frame(
            Time <- candle_data$Time[candle_first_size] - 60 * (aggregate - 1),
            Open <- candle_data$Open[1],
            High <- max(candle_data$High),
            Low <- min(candle_data$Low),
            Close <- candle_data$Close[candle_first_size],
            Volume <- sum(candle_data$Volume),
            Trades <- sum(candle_data$Trades),
            stringsAsFactors = FALSE)
          colnames(candle_first) <- colnames

          # Combine data
          aggregated_candles <- rbind(candle_first, aggregated_candles)
        }

        # If sparse last candle, calculate it
        if(candle_last_size != 0){
          # Set candles to be aggregated
          candle_data <- tail(temp_df, candle_last_size)

          # Aggregate last sparse candle
          candle_last <- data.frame(
            Time <- candle_data$Time[1],
            Open <- candle_data$Open[1],
            High <- max(candle_data$High),
            Low <- min(candle_data$Low),
            Close <- candle_data$Close[candle_last_size],
            Volume <- sum(candle_data$Volume),
            Trades <- sum(candle_data$Trades),
            stringsAsFactors = FALSE)
          colnames(candle_last) <- colnames

          # Combine data
          aggregated_candles <- rbind(aggregated_candles, candle_last)
        }
      }
    }

    # Make direction vector for plotting colors
    aggregated_candles$Direction <- ifelse(aggregated_candles$Close > aggregated_candles$Open, 1, 0)

    return(aggregated_candles)
  }

  names(data) <- pairs
  return(data)
}
```

B.2.4 | Factor Additon

After aggregating the candlesticks we add different factors to the data. We add the hour at which the candle starts, the direction of price movement in the candle, and three TA factors: The RSI, MACD, and ADX, for which we derive an indicator vector of whether we should buy or stay according to them. The function takes five arguments, `candlesticks` is the aggregated data using the code in Appendix B.2.3, `based_on` is the price data to base the TA factors on, `time_factor` is a boolean indicating whether or not to include the hour as factor, and `exclude_na` a boolean indicating whether or not to exclude the NA's produced by calculating the TA factors.

```r
# Add factors to candlesticks
AddTo_Candlesticks <- function(
  candlesticks, based_on = "Close", factors = FALSE, time_factor = FALSE, exclude_na = TRUE){

  # Set trading pairs to add factors to
  pairs <- names(candlesticks)

  # For each dataset
  data <- foreach(i = pairs) %do% {
    # Set temporary data frame
    temp_df <- candlesticks[[i]]
    if(factors == TRUE){
      # Calculate RSI, MACD, and ADX
      RSI <- TTR::RSI(temp_df[, based_on], n = 14, maType = "EMA", wilder = FALSE)
      MACD <-  TTR::MACD(temp_df[, based_on], maType = "EMA", nFast = 12, nSlow = 26, nSig = 9,
                         percent = TRUE, wilder = FALSE)
      ADX <-  TTR::ADX(temp_df[, c("High", "Low", "Close")], n = 14, maType = "EMA", wilder = FALSE)

      # Collect TA factors for plotting later on
      temp_factors <- as.data.frame(cbind(RSI, MACD, ADX))
      colnames(temp_factors) <- c("RSI", "MACD", "MACD_signal", "DIp", "DIn", "DX", "ADX")

      # Calculate trading signals based on TA factors
      RSI_action <- c(rep(0, 2), foreach(index = c(3:length(RSI)), .combine = c) %do% {
        action <-  0
        if(!is.na(RSI[index - 2])){
          if(RSI[index] >= 30 & RSI[index - 1] < 30) action <-  1
          if(RSI[index] >= 50 & RSI[index - 1] >= 50 & RSI[index - 2] < 50) action <-  1
        }
        return(action)
      })

      MACD_action <- c(rep(0, 4), foreach(index = c(5:nrow(MACD)), .combine = c) %do% {
        action <-  0
        if(!is.na(MACD[index - 4, 2])){
          check_0 <- MACD[index, 1] > MACD[index, 2]
          check_1 <- MACD[index - 1, 1] > MACD[index - 1, 2]
          check_2 <- MACD[index - 2, 1] > MACD[index - 2, 2]
          check_3 <- MACD[index - 3, 1] >= MACD[index - 3, 2]
          check_4 <- MACD[index - 4, 1] < MACD[index - 4, 2]
          if(check_0 & check_1 & check_2 & check_3 & check_4) action <-  1
        }
        return(action)
      })

      ADX_action <- c(rep(0, 3), foreach(index = c(4:nrow(ADX)), .combine = c) %do% {
        action <-  0
        if(!is.na(ADX[index - 3, 4])){
          check_movement <- ADX[index, 1] > ADX[index, 2]
          check_strength_0 <- ADX[index, 4] > 25
          check_strength_1 <- ADX[index, 4] > ADX[index - 1, 4]
          check_strength_2 <- ADX[index - 1, 4] > ADX[index - 2, 4]
          check_strength_3 <- ADX[index - 2, 4] > ADX[index - 3, 4]

          if(check_movement & check_strength_0){
            if(check_strength_1 | check_strength_2 | check_strength_3) action <- 1
          }
        }
        return(action)
      })

      # Collect TA signals
      temp_signals <- as.data.frame(cbind(RSI_action, MACD_action, ADX_action))
    }

    # Add timestamp as factor
    if(time_factor == TRUE) temp_df$Hour <- format(temp_df$Time, "%H")

    # Add factors to candlesticks
    if(factors == TRUE) temp_df <- cbind(temp_df, temp_signals, temp_factors)

    # Filter out candles with NA from MAs
    if(exclude_na) temp_df <- temp_df[complete.cases(temp_df),]

    return(temp_df)
  }

  # Return data
  names(data) <- pairs
  return(data)
}
```

B.2.5 | Classification

To classify the candlesticks into either buys or stays as described in Section 1.4.3, we use the function presented below. It takes five arguments, `candlesticks` is the data with added factors returned from the function in Appendix B.2.4, 'based_on' is the price to base the classification on (either close at current candle or open at next), `limit` is the desired percentage of profit, `stop` is the desired maximum percentage of loss, and `horizon` is the period of candles to base the classification on.

```r
Classify_Candlesticks <- function(
  candlesticks, based_on = "Close", limit = 0.01, stop = 0.02, horizon = 0){

  # Check inputs
  if(horizon == 0) {
    cat("Returning candlesticks input data as horizon = 0.")
    return(candlesticks)
  }
  if(limit < 0 | stop <= 0){
    stop("Limit and stop should be percentage in decimal. Stop will be negated automatically.")
  }

  pairs <- names(candlesticks)
  pairs_total <- length(pairs)

  # Classify the candles for each pair
  data <- foreach(i = pairs, .packages = c("foreach")) %do% {

    # Set temporary data frame
    temp_df <- candlesticks[[i]]
    candles_total <- nrow(temp_df) - horizon
    if(candles_total < 1){
      stop(paste0("Cannot classify over ", horizon, " candles, when only ", nrow(temp_df), "is given."))
    }

    # Classify the candles in one pair
    classes <- foreach(j = 1:candles_total, .combine = c) %do% {
      buy <- switch(based_on,
                    "Open" = temp_df$Open[j+1],
                    "Close" = temp_df$Close[j])

      # Set the desired leves for each observation and the highest and lowest value within the horizon
      goal_limit <- buy * (1 + limit)
      goal_stop <- buy * (1 - stop)
      highs <- temp_df$High[(j+1):(j+1+horizon)]
      lows <- temp_df$Low[(j+1):(j+1+horizon)]

      # Check if the limit and stop-limit is triggered
      tests_high <- highs >= goal_limit
      tests_low <- lows <= goal_stop

      # Check which limit is triggered first
      first_limit <- ifelse(any(tests_high == TRUE), min(which(tests_high == TRUE)), NA)
      first_stop <- ifelse(any(tests_low == TRUE), min(which(tests_low == TRUE)), NA)

      # Make the classification
      if(is.na(first_limit)){
        class <- 0
      } else {
        if(is.na(first_stop)){
          class <- 1
        } else {
          class <- ifelse(first_limit < first_stop, 1, 0)
        }
      }
      return(class)
    }

    # Set the remaining candles as stays
    Class <- c(classes, rep(0, horizon))
    classified_df <- cbind(Class, temp_df)
    return(classified_df)
  }

  names(data) <- pairs
  return(data)
}
```

B.2.6 | Differencing, Lagging, and Splitting

The final step of preparing the data is to difference, lag, and split it into training, validation, and test sets as described in Section 1.4.4. The function used to do this is presented below and takes seven arguments, `candlesticks` is the classified data returned from the function in Appendix B.2.5, `diff_value` is the order of differencing, `max_diff` is the maximum order of differecing used for all dataset, `lag` is the order of lag, `n_test` is the size of the test (and validation) set, `exclude` is a boolean indicating whether or not to exclude the direction factor, and `factors` is a boolean indicating whether or not to factors were added to the dataset.

```r
Split_Candlesticks <- function(
  candlesticks, diff_value = 0, max_diff = 0, lag = 0, n_test = 0.2, exclude = FALSE, factors = FALSE){

  # Check lag
  if(lag < 0) stop("Lag must be greater or equal to zero!")

  # Set pair names
  pairs <- names(candlesticks)

  data <- foreach(pair = pairs) %do% {
    temp_df <- candlesticks[[pair]]
    if(factors == TRUE){
      factors_df <- temp_df[, c("RSI", "MACD", "MACD_signal", "DIp", "DIn", "DX", "ADX")]
      non_lag_df <- temp_df[, c("Hour", "RSI_action", "MACD_action", "ADX_action")]
      colnames(non_lag_df) <- c("Hour_0", "RSI_action_0", "MACD_action_0", "ADX_action_0")
      temp_df <-
        temp_df[, c("Class", "Time", "Open", "High", "Low", "Close", "Volume", "Trades", "Direction")]
    }

    # Set number of observations in test set
    n_test <- round(nrow(temp_df) * n_test, 0)

    # Save start values to undiff to check actual profit/loss later
    start_values <- as.data.frame(rbind(
      temp_df[ifelse(lag + max_diff == 0, 1, lag + max_diff), c("Open", "High", "Low", "Close")],
      temp_df[(nrow(temp_df) - 2 * n_test - 2), c("Open", "High", "Low", "Close")],
      temp_df[(nrow(temp_df) - n_test - 1), c("Open", "High", "Low", "Close")]
    ))
    rownames(start_values) <- c("Train", "Validation", "Test")

    # Diff OHLC if diff_value > 0
    if(diff_value > 0){
      diff_series <- sapply(temp_df[, c("Open", "High", "Low", "Close")], diff, differences = diff_value)
      temp_df <- temp_df[-c(1:diff_value), ]
      temp_df[, c("Open", "High", "Low", "Close")] <- diff_series
      if(factors == TRUE){
        factors_df <- factors_df[-c(1:diff_value), ]
        non_lag_df <- non_lag_df[-c(1:diff_value), ]
      }
    }

    # If other diff values are used, cut datasets to same size
    if(diff_value < max_diff){
      temp_df <- temp_df[-c(1:(max_diff - diff_value)), ]
      if(factors == TRUE){
        factors_df <- factors_df[-c(1:(max_diff - diff_value)), ]
        non_lag_df <- non_lag_df[-c(1:(max_diff - diff_value)), ]
      }
    }

    # Check if lagged values exist
    if(lag >= nrow(temp_df)){
      stop(paste0("Can not use lag ", lag, " when ", i, " only has ", nrow(temp_df), " rows!"))
    }

    # Set response vector and initial design matrix
    if(lag == 0){
      y <- temp_df$Class
    } else {
      y <- temp_df$Class[-c(1:lag)]
      if(factors == TRUE){
        factors_df <- factors_df[-c(1:lag),]
        non_lag_df <- non_lag_df[-c(1:lag),]
      }
    }
    time <- temp_df$Time
    X_old <- temp_df[, -c(which(colnames(temp_df) %in% c("Class", "Time")))]


    # Exclude columns from design matrix if specified
    if(exclude == TRUE){
      X_old <- X_old[, -c(which(colnames(X_old) %in% c("Direction")))]
    }

    # Create colnames for lagged design matrix
    X_old_names <- colnames(X_old)
    X_new_names <- foreach(j = X_old_names, .combine = c) %do% {
      colnames <- foreach(k = 0:lag, .combine = c) %do% {
        paste0(j, "_", k)
      }
    }

    # Create lagged design matrix
    if(lag == 0){
      X <- X_old
    } else {
      X <- foreach(j = 1:ncol(X_old), .combine = cbind) %do% {
        lags <- as.data.frame(foreach(k = 0:lag, .combine = cbind) %do% {
          lagged_values <- lag(X_old[,j], k)
          cut_values <- lagged_values[c((lag + 1):length(lagged_values))]
          return(cut_values)
        })
      }
    }
    colnames(X) <- X_new_names

    if(factors == TRUE) X <- cbind(X, non_lag_df)

    # Training set
    time_train <- time[1:(length(y) - 2 * n_test - 2)]
    y_train <- y[1:(length(y) - 2 * n_test - 2)]
    X_train <- X[1:(length(y) - 2 * n_test - 2),]
    if(factors == TRUE) factors_train <- factors_df[1:(length(y) - 2 * n_test - 2),]

    # Validation set
    time_validation <- time[(length(y) - 2 * n_test - 1):(length(y) - n_test - 1)]
    y_validation <- y[(length(y) - 2 * n_test - 1):(length(y) - n_test - 1)]
    X_validation <- X[(length(y) - 2 * n_test - 1):(length(y) - n_test - 1),]
    if(factors == TRUE){
      factors_validation <- factors_df[(length(y) - 2 * n_test - 1):(length(y) - n_test - 1),]
    }

    # Test set
    time_test <- time[(length(y) - n_test):length(y)]
    y_test <- y[(length(y) - n_test):length(y)]
    X_test <- X[(length(y) - n_test):length(y),]
    if(factors == TRUE) factors_test <- factors_df[(length(y) - n_test):length(y),]

    # Add factors if true
    if(factors == TRUE){
      result <- list(time_train, y_train, X_train, factors_train,
                     time_validation, y_validation, X_validation, factors_validation,
                     time_test, y_test, X_test, factors_test, start_values)

      names(result) <- c("time_train", "y_train", "X_train", "factors_train",
                         "time_validation", "y_validation", "X_validation", "factors_validation",
                         "time_test", "y_test", "X_test", "factors_test", "Start_values")
    } else {
      result <- list(time_train, y_train, X_train,
                     time_validation, y_validation, X_validation,
                     time_test, y_test, X_test, start_values)

      names(result) <- c("time_train", "y_train", "X_train",
                         "time_validation", "y_validation", "X_validation",
                         "time_test", "y_test", "X_test", "Start_values")
    }
    return(result)
  }

  names(data) <- pairs
  return(data)
}
```

B.2.7 | Calculating Profits

The function used to do this is presented below and takes eight arguments, `data` is the dataset for which the predictions are made upon, `predicted` is a vector of predicted classes, `set` is the name of the set the predictions are performed on, `horizon` is the number of candles to calculate the profit on, `ignore_stops` is a boolean indicating whether or not to ignore the stops used in the data, `PL` is a boolean indicating whether to calculate the profits and losses or create a detailed summary, `fee` is the fee percentage to use, and `parameters` is the parameters used for calculating the potential profits when used for multiple datasets.

```r
Calculate_Profit <- function(
  data, predicted, set, horizon = 24, ignore_stops = FALSE, PL = FALSE, fee = 0.001, parameters){

  # Set data to be used for calculating profit and loss
  if(set == "scouting"){
    X <- rbind(data$X_train, data$X_validation)
    y <- c(data$y_train, data$y_validation)
    predicted <- y
    set <- "train"
    start_values = data$Start_values
  } else {
    X <- data$Data[[paste0("X_", set)]]
    y <- data$Data[[paste0("y_", set)]]
    start_values = data$Data$Start_values
    parameters = data$Parameters
  }

  # If data was differenced undiff it
  if(parameters$diff_value > 0){
    closes <- diffinv(X$Close_0, xi = start_values[capitalize(set), "Close"])[-1]
    highs <- diffinv(X$High_0, xi = start_values[capitalize(set), "High"])[-1]
    lows <- diffinv(X$Low_0, xi = start_values[capitalize(set), "Low"])[-1]
    closes <- diffinv(X$Close_0, xi = start_values[capitalize(set), "Close"])[-1]
  } else {
    opens <- X$Open_0
    highs <- X$High_0
    lows <- X$Low_0
    closes <- X$Close_0
  }

  # Figure out which buys were good and which were bad
  true_buys <- intersect(which(predicted == 1), which(y == 1))
  other_buys <- intersect(which(predicted == 1), which(y == 0))
  true_stays <- intersect(which(predicted == 0), which(y == 0))
  other_stays <- intersect(which(predicted == 0), which(y == 1))

  # Calculate loss for each wrong buy
  if(length(other_buys) == 0){
    losses <- 0
  } else {

    losses <- foreach(buy = other_buys, .combine = c) %do% {
      buy_price <- closes[buy]

      # If ignore stops, just sell at horizon
      if(ignore_stops == TRUE){
        if(length(closes[(buy + 1):length(closes)]) > horizon){
          loss <- (closes[(buy + 1):length(closes)][horizon] - buy_price) / buy_price
        } else {
          loss <- (tail(closes, 1) - buy_price) / buy_price
        }

        # If not ignoring stops, sell at stop percentages
      } else {
        stop_price <- buy_price * (1 - parameters$stop)

        triggered_at <- detect_index(lows[(buy + 1):length(lows)], function(x) x <= stop_price)

        if(triggered_at > 0 & triggered_at <= horizon){
          loss <- -parameters$stop
        } else if(length(closes[(buy + 1):length(closes)]) > horizon){
          loss <- (closes[(buy + 1):length(closes)][horizon] - buy_price) / buy_price
        } else{
          loss <- (tail(closes, 1) - buy_price) / buy_price
        }
      }
      return(loss)
    }
  }

  # If PL == TRUE calculate cumulative profit and loss
  if(PL == TRUE){
    PL <- rep(0, length(predicted))
    PL[true_buys] <- parameters$limit - fee - (1 + parameters$limit) * fee
    PL[other_buys] <- losses - fee - (1 + losses) * fee
    PL[true_stays] <- 0
    PL[other_stays] <- 0

    result <- PL
  } else {

    # Collect results for table
    # Number of buys
    n_buys <- length(true_buys) + length(other_buys)
    n_true_buys <- length(true_buys)
    n_false_buys <- length(other_buys)
    n_losses <- length(which(losses < 0))

    # Number of stays
    n_stays <- length(true_stays) + length(other_stays)
    n_true_stays <- length(true_stays)
    n_false_stays <- length(other_stays)

    # Fees and profits
    fees <- (n_buys * fee) + (n_true_buys * (1 + parameters$limit) * fee) +
      (n_false_buys * (1 + mean(losses)) * fee)
    profit <- (length(true_buys) * parameters$limit) + sum(losses) - fees

    result <- as.data.frame(
      cbind(n_buys, n_true_buys, n_false_buys,
            n_stays, n_true_stays, n_false_stays,
            n_losses, fees, profit)
    )
  }

  return(result)
}
```

B.3 | IMDb Example

In this appendix we present the code needed to reproduce the IMDb example shown in Chapter 5.

B.3.1 | Setup

The following code shows the data processing needed to reproduce the IMDb example. First we import the dataset, which is contained in the Keras R-package, the `num_words` argument determines the amount of words to use, in this case only the 10000 most popular. Subsequently we extract the training and test data. The raw data is contained within lists and needs to be in a matrix format in order to use it for modelling. We reformat the data into matrices in lines 14-27. To monitor the generalization performance of the models during training we extract some obeservations for a validation set and keep the rest for the training set.

```
 1 # First load the packages used
 2 library(keras)
 3 library(xgboost)
 4 library(randomForest)
 5 
 6 # Then load the dataset
 7 imdb <- dataset_imdb(num_words = 10000)
 8 train_data <- imdb$train$x
 9 train_labels <- imdb$train$y
10 test_data <- imdb$test$x
11 test_labels <- imdb$test$y
12 
13 
14 # The following function formats data into a mtrix
15 vectorize_sequence <- function(sequences, dimension = 10000) {
16   results <- matrix(0, nrow = length(sequences), ncol = dimension)
17   for (i in 1:length(sequences))
18     results[i, sequences[[i]]] <- 1
19   return(results)
20 }
21 
22 # Format data
23 x_train <- vectorize_sequence(train_data)
24 x_test <- vectorize_sequence(test_data)
25 
26 y_train <- as.numeric(train_labels)
27 y_test <- as.numeric(test_labels)
28 
29 # To monitor generalization during traning we set aside a validation set
30 val_indices <- 1:10000
31 
32 x_val <- x_train[val_indices,]
33 partial_x_train <- x_train[-val_indices,]
34 
35 y_val <- y_train[val_indices]
36 partial_y_train <- y_train[-val_indices]
```

B.3.2 | Neural Networks

We start off by training a neural network on the processed IMDb data by first defining the topography of network. Subsequently we define the compile options, which define the optimization algorithm, loss function to be minimized, and metric to monitor during training, in addition to the training error. In lines 17-24 we train the models, which automatically generates a plot showing the evolution of the different performance metrics as the number of epochs range from 1 to 20. Based on the performance metrics shown during training we decide to reduce the number of epochs to 5, then retrain the model on the original training set, and use this model to perform predictions on the test set. The performance on the test is evaluated in line 48.

```r
# Define the model
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

# We have defined the topography of the model now we compile it
# which means defining optimization, loss function, and metrics to watch during training
model %>%
  compile(optimizer = "rmsprop",
          loss = "binary_crossentropy",
          metrics = c("accuracy")
  )

# Now we are set and can train our model
# This call also plots validation and trainig accuracy, and loss as a function of epochs
history <- model %>%
  fit(partial_x_train,
      partial_y_train,
      epochs = 20,
      batch_size = 512,
      validation_data = list(x_val, y_val),
      plot = TRUE
  )

# As evident from the plots 20 epochs is overfitting so we fit a new model using only 4 epochs
model1 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model1 %>%
  compile(optimizer = "rmsprop",
          loss = "binary_crossentropy",
          metrics = c("accuracy")
  )

# Now we use the full dataset
history1 <- model1 %>%
  fit(x_train,
      y_train,
      epochs = 5,
      batch_size = 512,
      validation_data = list(x_val, y_val)
  )

# Now use the sexy Keras library to evaluate how well we did on the test data
results <- model1 %>% evaluate(x_test, y_test)
results # This naive approach yielded accuracy of 87% might vary with randomness

# Let us generate predictions form some data yielding probabilities of reviews being positive
predictions <- model1 %>% predict(x_test)
```

B.3.3 | Gradient Boosting

Here we apply gradient boosting to the IMDb example by first transforming the processed training and validation data into `xgb.DMatrix` format and then fit the model. In lines 17-21 we extract the training and validation error, which we use to asses whether or not we are overfitting. In lines 24-26 we extract the predicted probabilities and calculate accuracy. Since we do not see any signs of overfitting, we retrain the model on the full dataset and subsequently calculate the accuracy.

```r
# Prepare the data for XGB
dtrain <- xgb.DMatrix(data = partial_x_train, label = partial_y_train)
dval <- xgb.DMatrix(data = x_val, label = y_val)
watchlist <- list(train = dtrain, test = dval)

# Run the model
model <- xgb.train(data = dtrain,
                    max_depth = 3,
                    eta = 0.3,
                    nthread = 4,
                    nrounds = 200,
                    watchlist = watchlist,
                    objective = "binary:logistic",
                    lambda = 0)

# Get training and validation errors
train_err <- data.frame(err = model$evaluation_log$train_error)
train_err$iter <- 1:length(train_err$err)

val_err <- data.frame(err = model$evaluation_log$test_error)
val_err$iter <- 1:length(val_err$err)

# Get true classification rate
validation_probabilties <- predict(bst, x_val)
validation_prediction <- (validation_probabilties > 0.5)
sum(validation_prediction == y_val)/length(y_val)

# Full data
dtrain <- xgb.DMatrix(data = x_train, label = y_train)
dval <- xgb.DMatrix(data = x_test, label = y_test)
watchlist <- list(train = dtrain, test = dval)

model1 <- xgb.train(data=dtrain,
                   max_depth = 3,
                   eta = 0.3,
                   nthread = 4,
                   nrounds = 200,
                   watchlist = watchlist,
                   objective = "binary:logistic",
                   lambda = 0)

# Get true classification rate for full data set
validation_probabilties <- predict(model1, x_test)
validation_prediction <-(validation_probabilties > 0.5)
sum(validation_prediction == y_test)/length(y_test)
model$evaluation_log$test_error[200]
```

B.3.4 | Random Forest

Since we do not perform any model configuration using random forests we simply start off by fitting the model on the full dataset. We have however reduced the default number of trees, from 500 to 250, to reduce the training time, which is still very long. In lines 9-13 we extract the training and OOB error and subsequently calculate the classification accuracy on the test set.

```r
 1 # Grow the forest
 2 model <- randomForest(x = x_train,
 3                       y = as.factor(y_train),
 4                       ytest = as.factor(y_test),
 5                       xtest = x_test,
 6                       do.trace = TRUE,
 7                       ntree = 250)
 8
 9 # Get OOB and Test errors
10 test_error <- data.frame(err = model$test$err.rate[,1])
11 oob_error <- data.frame(err = model$err.rate[,1])
12 test_error$iter <- 1:250
13 oob_error$iter <- 1:250
14
15 # Get true classification rate
16 model$test$err.rate[250,1]
17 test_probabilities <- as.vector(model$test$votes[,2])
18 test_predictions <- (test_probabilities > 0.5)
19 sum(y_test == test_predictions)/length(y_test)
```

C | Trade Plots

Similar to the model performance plots for the BTC-USDT trading pair seen in Figures 8.3 and 8.6, Figures C.1-C.5 in this appendix show the model performance on the five other trading pairs: ETH-USDT, BNB-USDT, NEO-USDT, LTC-USDT, and BCC-USDT. The model performance plot for each model covers the same period from May 1st, 2018 at 01:00 to May 16th at 09:59 and consists of three plots.

- The **top** plot shows the price movement of the trading pair in the period, charted as candles.
- The **middle** plot shows the models' classificaitons of buys and stays, where the 'True' (blue) are the correctly classified buys resulting in a 2% profit, the 'Profit'(green) are the wrongly classified buys that resulted in a profit, and the 'Loss' (red) are the wrongly classified buys that resulted in a loss.
- The **bottom** plot shows the cumulative returns of the models in the period.

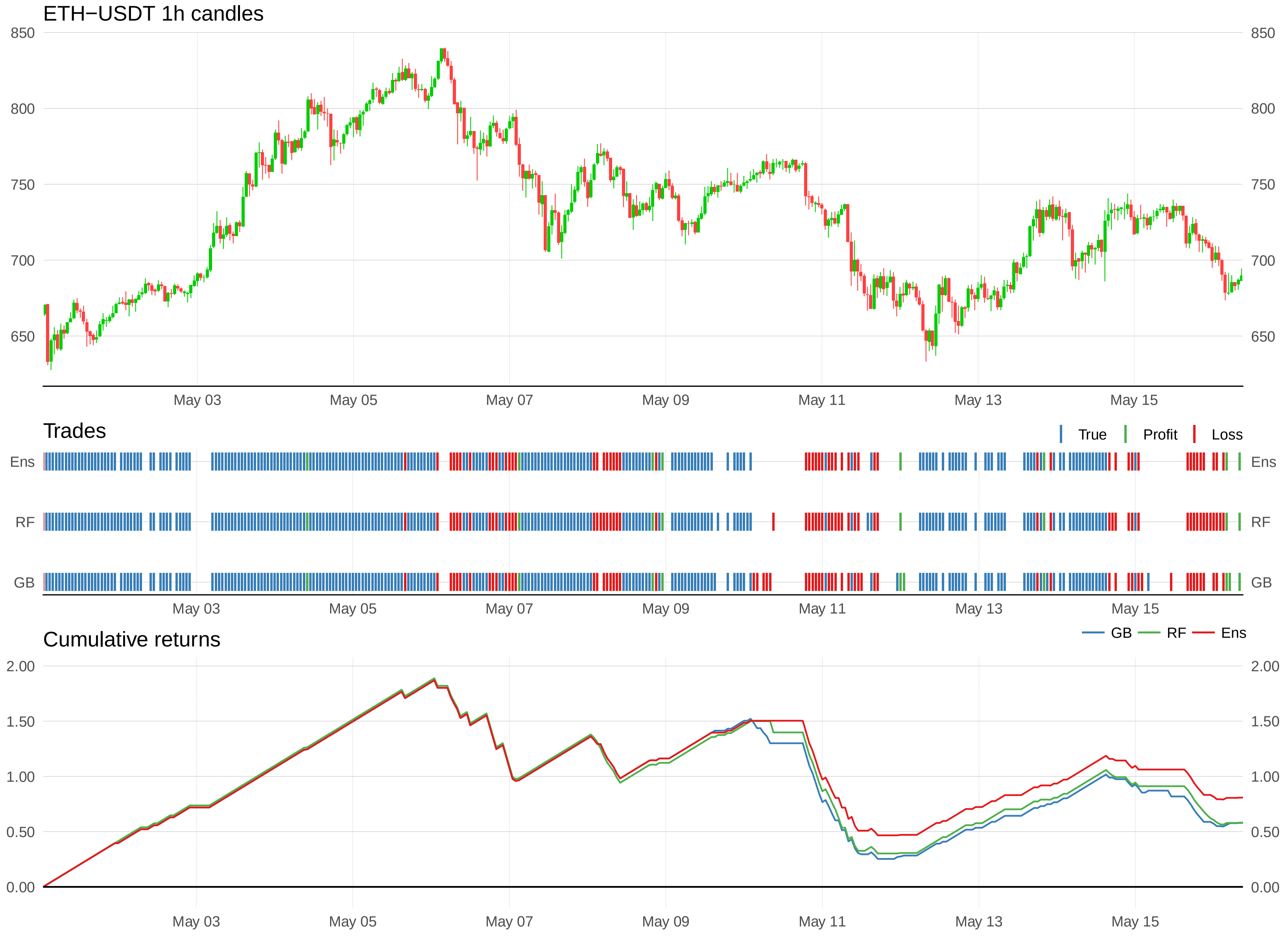


Figure C.1: Model performance on the ETH-USDT 1h candles in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59. **Top:** The candles in the period. **Middle:** The models' classifications of buys and stays. True (blue) are correctly classified buys resulting in a 2% profit, Profit (green) are wrongly classified buys that resulted in a profit, and Loss (red) are wrongly classified buys resulting in a loss. **Bottom:** The cumulative returns of the models through the period.

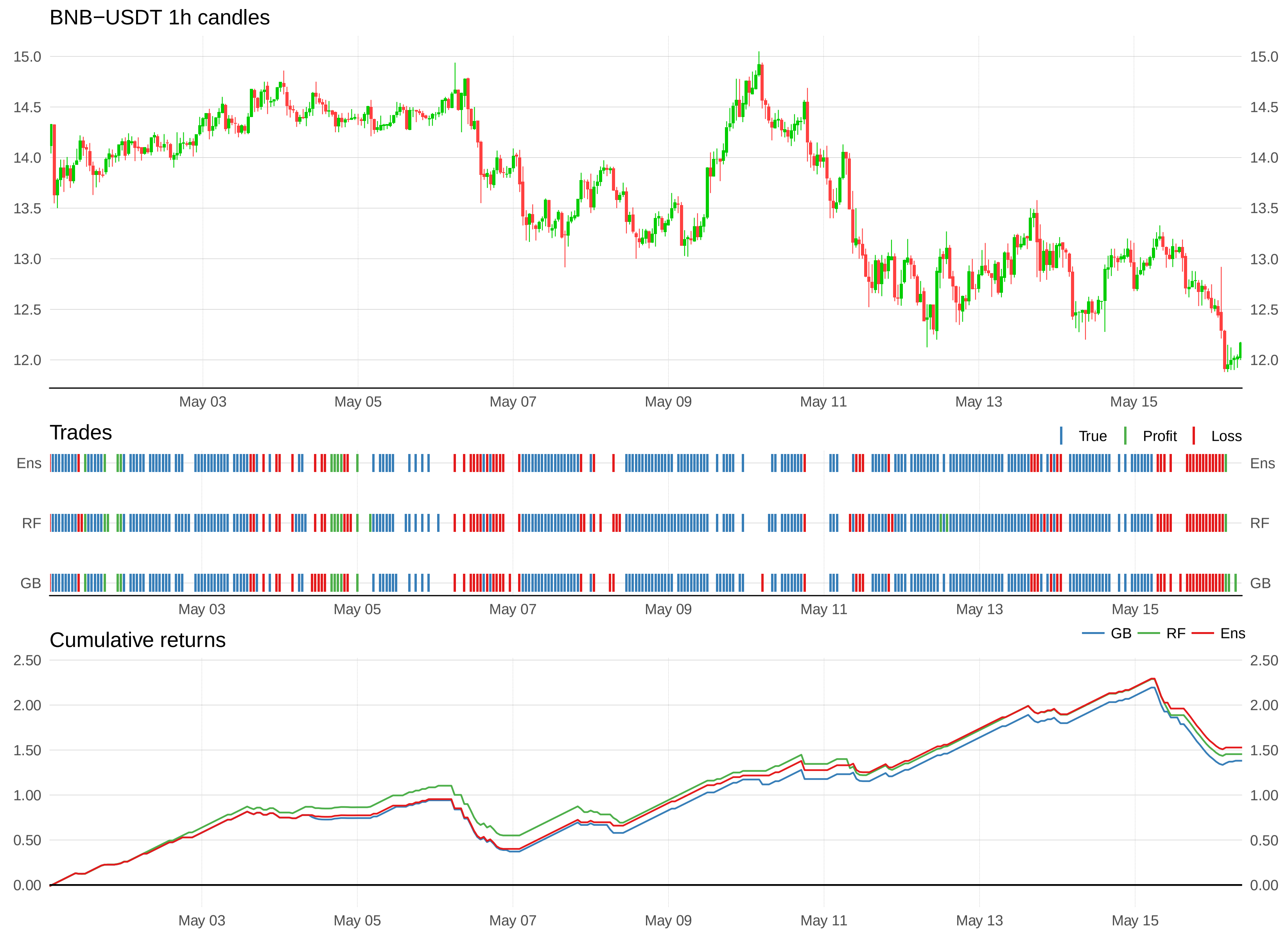


Figure C.2: Model performance on the BNB-USDT 1h candles in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59. **Top:** The candles in the period. **Middle:** The models' classification of buys and stays. True (blue) are correctly classified buys resulting in a 2% profit, Profit (green) are wrongly classified buys that resulted in a profit, and Loss (red) are wrongly classified buys resulting in a loss. **Bottom:** The cumulative returns of the models through the period.

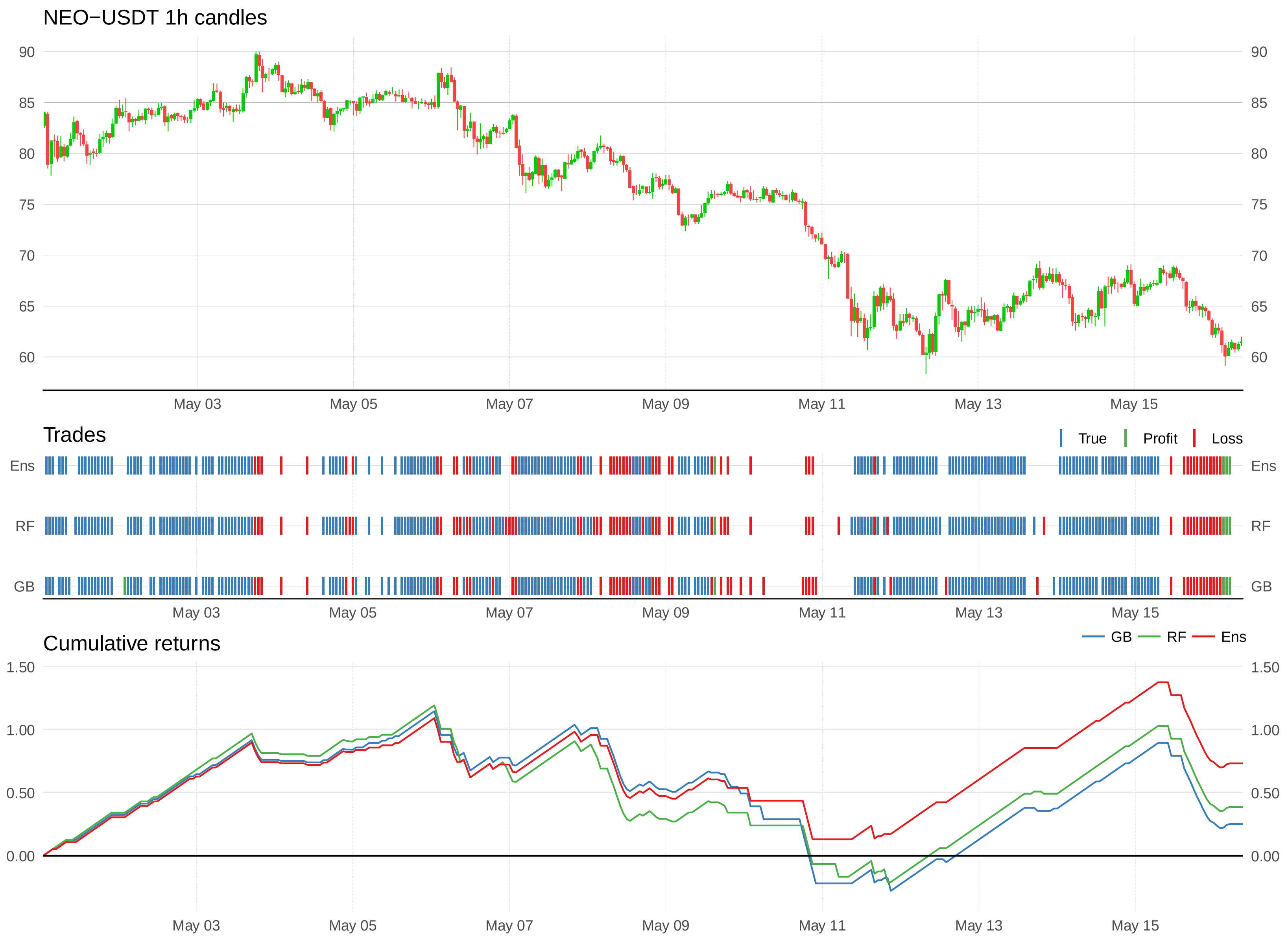


Figure C.3: Model performance on the NEO-USDT 1h candles in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59. **Top:** The candles in the period. **Middle:** The models' classification of buys and stays. True (blue) are correctly classified buys resulting in a 2% profit, Profit (green) are wrongly classified buys that resulted in a profit, and Loss (red) are wrongly classified buys resulting in a loss. **Bottom:** The cumulative returns of the models through the period.

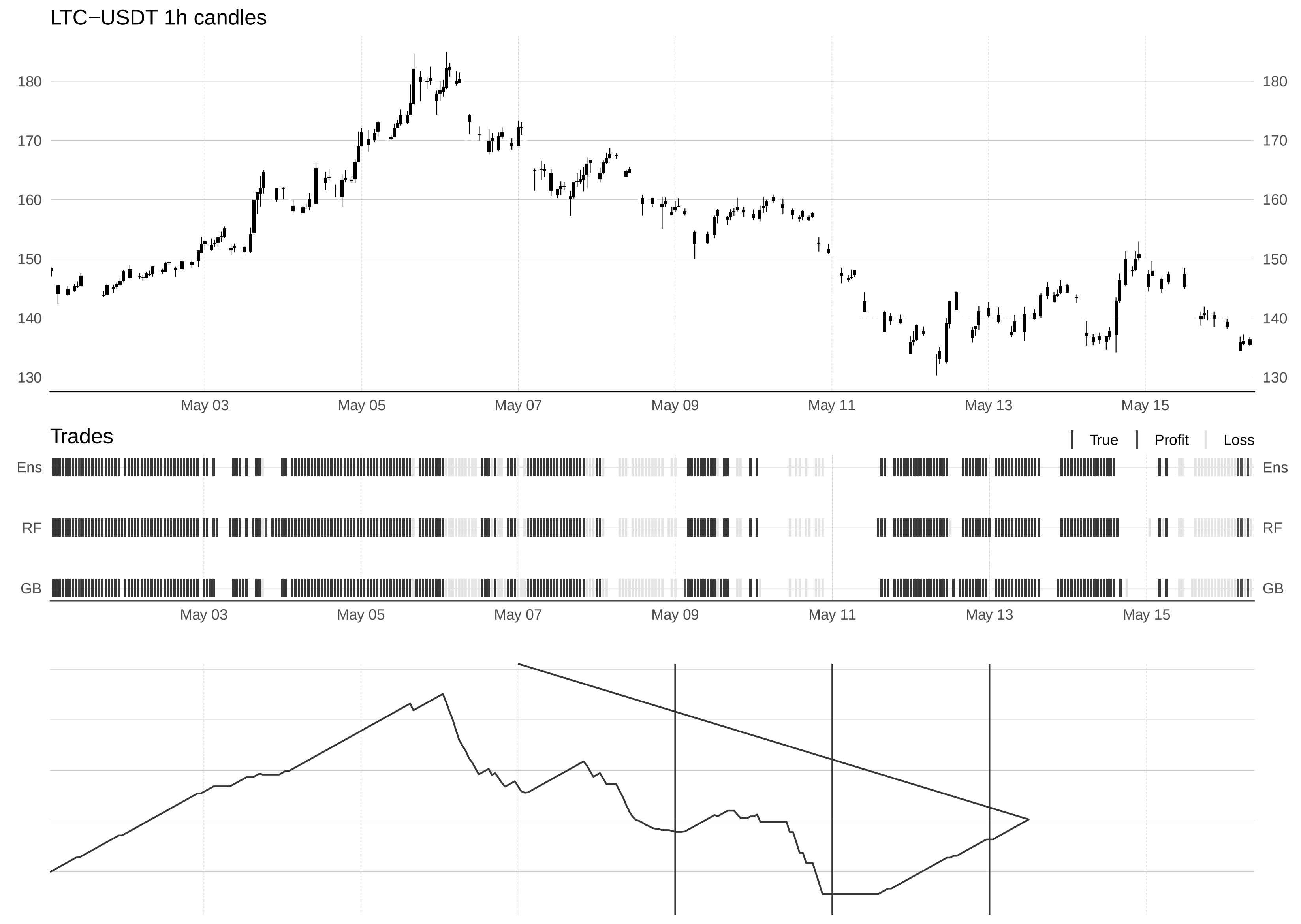


Figure C.4: Model performance on the LTC-USDT 1h candles in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59. **Top:** The candles in the period. **Middle:** The models' classification of buys and stays. True (blue) are correctly classified buys resulting in a 2% profit, Profit (green) are wrongly classified buys that resulted in a profit, and Loss (red) are wrongly classified buys resulting in a loss. **Bottom:** The cumulative returns of the models through the period.

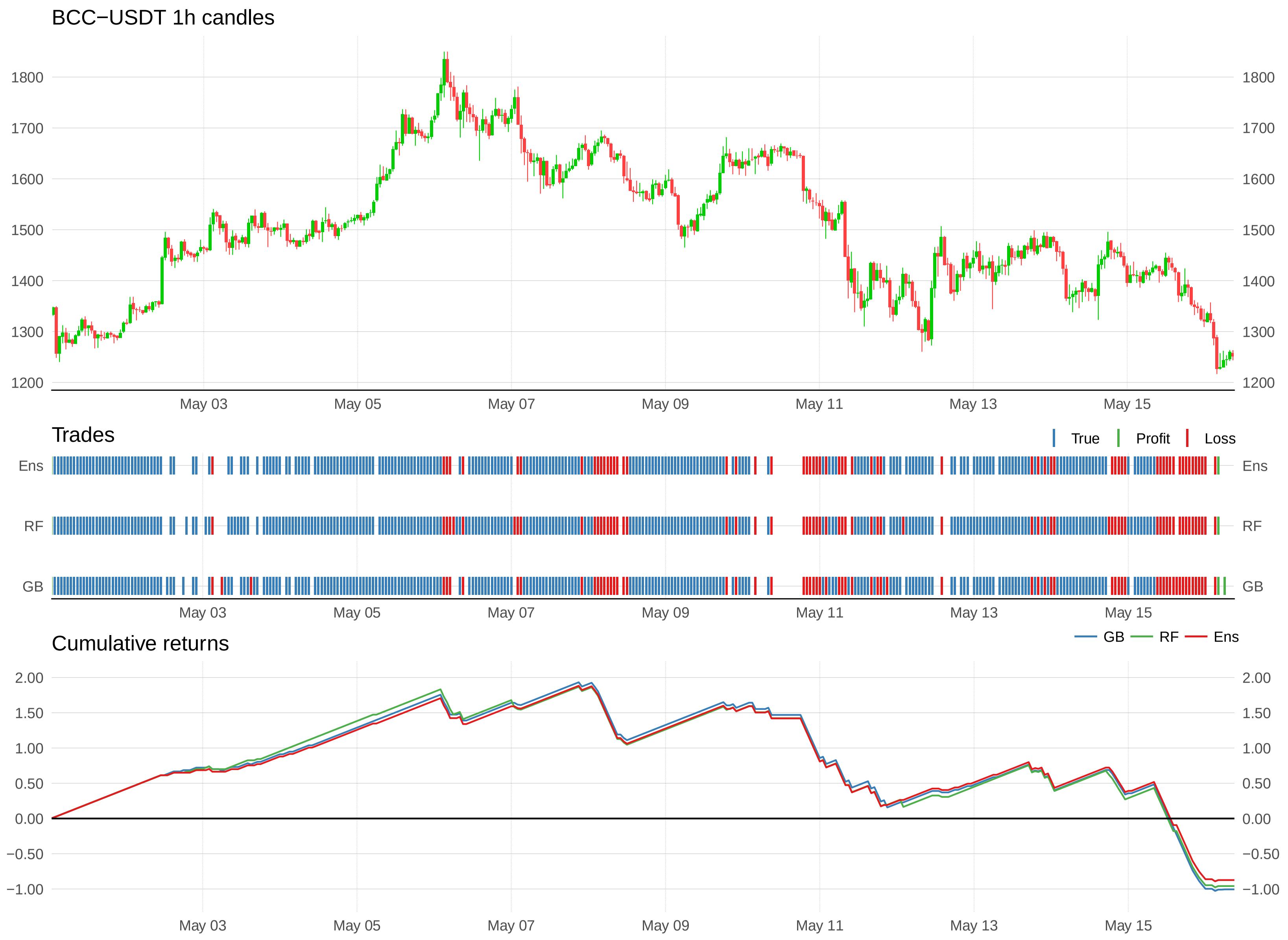


Figure C.5: Model performance on the BCC-USDT 1h candles in the period from May 1st, 2018 at 01:00 to May 16th, 2018 at 09:59. **Top:** The candles in the period. **Middle:** The models' classification of buys and stays. True (blue) are correctly classified buys resulting in a 2% profit, Profit (green) are wrongly classified buys that resulted in a profit, and Loss (red) are wrongly classified buys resulting in a loss. **Bottom:** The cumulative returns of the models through the period.

Bibliography

A. Agresti. *An Introduction to Categorical Data Analysis*. Wiley Series in Probability and Statistics. Wiley, 2007. ISBN 9780470114742. URL `https://books.google.dk/books?id=OG9EqwdOFh4C`.

Binance API. Binance crypto exchange api documentation, 2018. URL `https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md`.

Binance. Binance crypto exchange, 2018a. URL `https://www.binance.com/`.

Binance. Binance crypto exchange advanced interface, 2018b. URL `https://www.binance.com/tradeDetail.html`.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Coinbase. Coinbase crypto exchange, 2018. URL `https://www.coinbase.com/`.

CoinMarketCap. Coinmarketcap cryptocurrency tracker, 2018. URL `https://coinmarketcap.com/`.

Binance Fee. Binance crypto exchange fees, 2018. URL `https://support.binance.com/hc/en-us/articles/115000429332-Fee-Structure-on-Binance`.

Joseph J. Allaire Francois Chollet. *Deep Learning With R*. Manning Publications CO., New York, NY, USA, 2018.

Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent, 2009.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine, April 2001. URL `https://statweb.stanford.edu/~jhf/ftp/trebst.pdf`.

Blaise Hanczar, Jianping Hua, Chao Sima, John Weinstein, Michael Bittner, and Edward R. Dougherty. Small-sample precision of roc-related estimates. *Bioinformatics*, 26(6):822–830, 2010. doi: 10.1093/bioinformatics/btq037. URL `http://dx.doi.org/10.1093/bioinformatics/btq037`.

Trevor Hastie and Junyang Qian. Glmnet vignette, 2016. R package version 1.9-16.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

Jorge M. Lobo, Alberto Jiménez-Valverde, and Raimundo Real. Auc: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography*, 17(2):145–151, 2007. doi: 10.1111/j.1466-8238.2007.00358.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1466-8238.2007.00358.x`.

Joshua Ulrich. *TTR: Technical Trading Rules*, 2017. URL `https://CRAN.R-project.org/package=TTR`. R package version 0.23-2.