Master Thesis

University of Bonn

Master of Science in Economics

Identifying Arbitrage in Cryptomarkets with Algorithmic Trading:

A Machine Learning Approach

Submitted in August 2020 by

Raphael Redmer

Matriculation Number: 2645976

Supervisor: Prof. Dr. Joachim Freyberger

Abstract

In this master thesis, I used various machine learning methods to predict price movements based on minute-binned OHLC price data from a range crypto currencies obtained from the crypto exchange Bitfinex [22]. The probability signals from these models are then used to inform the trading algorithm, which decides to open up either a short or long position for 120min. The models are trained over a time period of 01.01.2019 to 14.09.2019, and then subsequently tested on data ranging from 01.1.2019 to 31.12.2019. Then, I proceeded to analyse the results in order to determine whether these models can be used to identify arbitrage opportunites and if the first efficient market hypothesis is violated. I found that it is not consistently possible in this experimental setup to identify arbitrage and generate superior profits. Even with fine tuning the probability threshold, the best model only managed to achieve 15,1% return over the test period. However, this is outmatched by a simple buy-and-hold equal-weight portfolio which achieved a return of 42%. Without fine tuning the probability threshold, the models consistently generate negative returns.

Contents

1 Introduction

In the realm of finance literature, it is well known that financial time series are notoriously difficult to predict, primarily driven by the high degree of noise [12]. Moreover, the generally accepted weak efficient market hypothesis provides a theoretical framework which encompasses this phenomenon [10]. The weak form states that current asset prices reflect all the information of past prices and that no form of analysis can be effectively utilized to consistently support investors in making trading decisions, thus eliminating arbitrage solely based on price data. Therefore, also machine learning algorithms which were trained on price data should not be able to do so according to this hypothesis. In order to test this hypothesis, I trained multiple machine learning models on price data with different feature and target specifications to predict price movements. I chose a classification instead of a regression problem, as the literature suggests that it performs better for predicting financial market data ([27], [9]). Then, I used the estimated probabilities from the models in a backtest to simulate actual trading with a trading algorithm. Model training and backtest are conducted on minute-binned OHLC-data of cryptocurrency coins from the Bitfinex exchange. The reason for choosing cryptocurrency assets is the fact that they have remained fairly unregulated by governmental institutions ([8], [20]). Therefore, this asset class and its exchanges are more in line with the underlying assumption of the efficient market hypothesis such as perfect markets, thus I expected that arbitrage opportunities only rarely occur. This expectation is further supported by low entry barriers and transaction costs [22].

This work extends the existing literature by

- including volumes traded in the respective minute-bin,
- comparing the effects of different durations,
- explaining in detail how transaction costs got incorporated,
- analysing how changing the probability threshold affects returns and
- improving the trading algorithm by not making it automatically close an active position, regardless of probability signal.

2 Literature Review

The authors of [42] predicted price changes with a Bayesian regression model of Bitcoin during a six month period in 2014. The model performed exceptionally well with a return of 89 percent and a Sharpe ratio of 4.10 during a period of 50 trading days. However, no transaction costs were not incorporated and they assumed perfect liquidity. [43] used a momentum strategy on the U.S. CRSP stock universe ranging from 1965 to 2009. This strategy was based on deep neural network classifiers to estimate the probability for each stock to outperform the cross-sectional median return of all stocks in the holding month $t + 1$. [7] achieved a relatively high classification accuracy of 73 percent with five-minute binned return data. However, they did not take microstructural market effects into account which is essential when dealing with high-frequency data. [31] trained a random forests on U.S. CRSP data ranging from 1968 to 2012 to incorporate it into a trading strategy. [11] trained a random forest on the lagged returns of 40 cryptocurrency coins to predict whether a coin outperforms the cross-sectional median of 40 coins over a forecast period of 120 min. They enter a long position with the top-3 coins and a long position with the bottom-3 coins in terms of prediction probability. After 120 min, they reverse the respective positions. During the out-of-sample period of their backtest, ranging from 18 June 2018 to 17 September 2018, and after more than 100,000 trades, they find statistically and economically significant returns of 7.1 bps per day, after transaction costs of 15 bps per half-turn. [26] analysed the effectiveness of deep neural networks, gradientboosted trees, Random Forests, and a combination of these methods for identifying statistical arbitrage. Each model got trained on lagged returns of all stocks in the S&P 500. Daily one-day-ahead trading signals are generated based on the probability forecast of a stock to outperform the market.

3 Data and Software

3.1 Data

In this experimental setup, I used minute-binned OHLC data of crypto/USD-pairs obtained from the cryptocurrency exchange Bitfinex ranging from 01.01.2019 to 31.12.2019 via its official API. For each minute-bin, *Open*, *High*, *Low*, *Close*, *Volume* and *Timestamp* data got collected. *High* and *Low* denote the highest and lowest price respectively that was traded within this timeframe. *Open* and *Close* denote the first and last traded price. *Volume* denotes the total volume traded within the respective minute-bin. *Timestamp* denotes the point in time for each minute-bin as a UNIX-Timestamp, i.e., is the number of seconds that have passed since 01.01.1970.

Even though Bitfinex is the largest exchange for cryptocurrency with a daily trading volume of roughly 111 million USD and 155 different Dollar-tradable coins [22], for most coins, the trading frequency is so low such that many crypto/USD-pairs have a considerable amount of minute-bins in which no volume was traded (see figure 1).

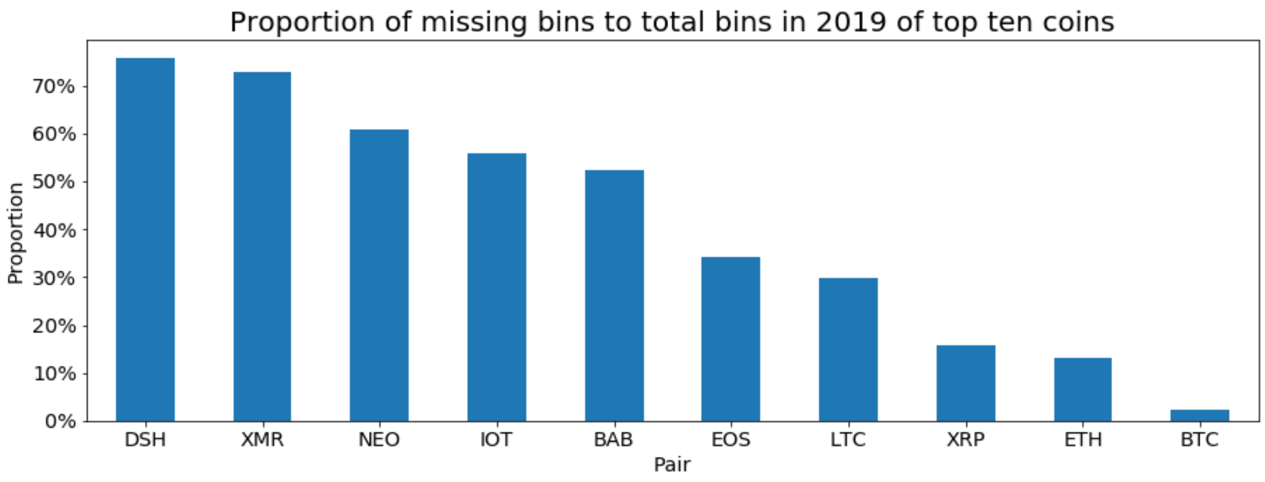


Figure 1: This figure illustrates the proportion of missing minute-bins to total number of minute-bins in 2019 for the top ten coins.

In case of a crypto-pair having no volume for a particular minute, the API leaves out this bin when requesting its data resulting in missing bins. This issue got resolved by propagating price values from the last active minute-bin and setting the volume to zero. Further, I restricted the number of crypto/USD-pairs to the top ten pairs by market capitalization [5]. In addition, I decided to only take data from 01.01.2019 to 31.12.2019, since 2019 was the most active year in terms of trading frequency for most coins. Thus, the resulting data set contains roughly 10 $\times$ 365 $\times$ 24 $\times$ 60 $=$ 5.256.000 rows.

3.2 Software

The programming language used for conducting this study is Python 3.7 [37]. For data preparation and feature engineering, I used Pandas and numpy ([35], [33]). Data Visualization was done via Maplotllib [34]. I used the respective Scikit-learn implementation of the Machine Learning models used in this master thesis [41].

4 Methodology

Similarly to [11] and [26], the methodology of this paper consists of the following steps:

1. The entire data set is split into training, validation and trading set
2. The respective features (explanatory variables) and targets (dependent variables) are created
3. Each model is trained on the training and validation set
4. Conduct out-of-sample predictions and backtest trading on the trading set for each model
5. Evaluate its accuracy and trading-performance on the trading set respectively
6. Go to Step 2, and repeat the same steps for a different feature and target specification

4.1 Training and Trading Set

In this application to minute-binned data, the test set, i.e., trading set contains all observations from 01.11.2019 to 31.12.2019. The training set ranges from 01.01.2019 to 14.09.2019, and the remaining 15.09.2019 to 31.10.2019 is reserved for validation. I decided against the usual k-fold cross-validation approach in order to emphasize the importance of future observation for the model, since its performance only gets evaluated on the future trading set. The daily cumulative short returns in the trading period of the top ten coins can be seen below in figure 2. The returns in the trading period exhibit a downward trajectory across all coins, thus the positive short returns.

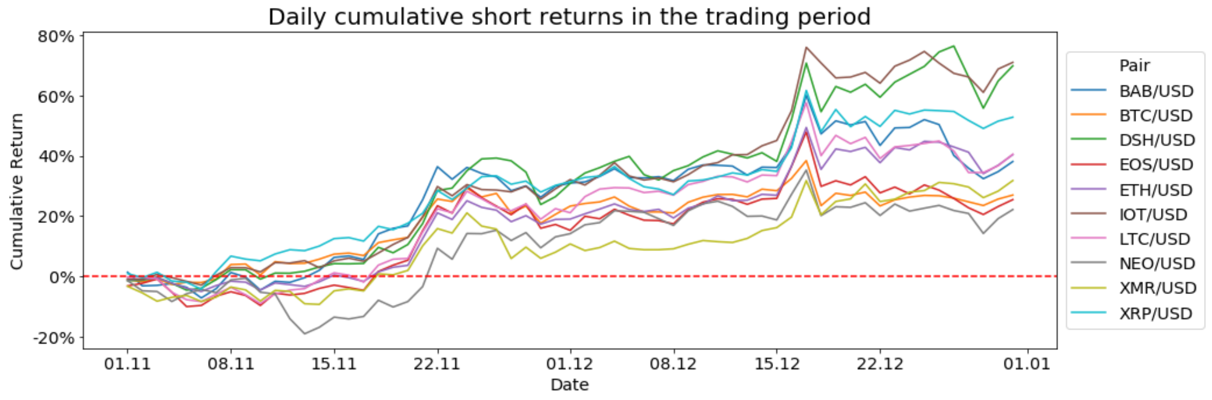


Figure 2: This figure illustrates daily cumulative short returns in the trading period for the top ten coins selection.

4.2 Feature and Target Generation

Broadly following [43], the feature space gets generated as follows.

Let $P^c = (P^c_t)_{t \in T}$ denote the price process of coin/USD-pair c, with $c \in \{1, ..., n\}$. The price itself is defined as the average between *Open* and *Close*.

The following features and targets are generated from the data set:

Features: **Returns:** Let $R^c_{t,t-m}$ be the simple return for coin c over m periods defined as

$$R^c_{t,t-m} = \frac{P^c_t}{P^c_{t-m}} - 1 \tag{1}$$

Volumes: Let V^c_t be the traded volume for coin c in minute-bin t scaled by Quantile-Transformer fitted separately for each coin only on bins having traded volume.

Target: Let $Y^c_{t+d,t}$ be a binary response variable for each coin c and the duration of the forecast interval $d = 120, 240$. It assumes value 1 (class *up*) if its future d min return $R^c_{t+d+1,t+1}$ is greater than its cross-sectional median across all pairs $(R^c_{t+d+1,t+1})^C_{c=1}$, else -1 (class *down*). Instead of just using the simple return $R^c_{t+d+1,t+1}$ as in [11], I included an additional condition, which demands that $V^c_{t+d+1} > 0$ for realizing the feature return. If not skip bins until you reach a bin t^* in which $V^c_{t^*} > 0$, then realize return as in equation 1.

The reason for this further restriction is to make the training of the model more similar to the trading decisions in the backtest, since only trades are allowed to be executed in a bin if any volume was traded for the respective coin. I decided for the inclusion of volume such that the model has a measure for taking trading activity into account without breaking vital assumptions needed for testing the 1. Efficient Market Hypothesis ([29], [10]). In addition, the volume got scaled for each coin in order to make the measure more comparable across coins, since a single universal model is trained for each of the selected coins. Further, the Quantile-Transformation handles outliers and restricts the features to an interval ranging from 0 to 1.

Let τ denote the transaction cost per trade in basis points (bps) proportional to the traded amount. For sake of simplicity, assume that there is no differentiation between maker and taker transaction costs, thus having equal transaction cost $\tau = \tau_{taker} = \tau_{maker}$. Incorporating costs τ leads to following expression for return $R^c_{t,t-m}$ in long a position,

$$\begin{aligned} R^{c,long}_{t,t-m} &= \frac{P^c_t(1-\tau) - P^c_{t-m}(1+\tau)}{P^c_{t-m}(1+\tau)} \\ &= \frac{P^c_t}{P^c_{t-m}} \frac{1-\tau}{1+\tau} - 1, \end{aligned} \tag{2}$$

and analogously for the short position,

$$R^{c,short}_{t,t-m} = \frac{P^c_{t-m}}{P^c_t} \frac{1-\tau}{1+\tau} - 1 \tag{3}$$

4.3 Model Training

As explained in chapter 4.1, I constructed a training set ranging from 01.01.2019 to 14.09.2019, a validation set ranging from 15.09.2019 to 31.10.2019, and a trading set ranging from 01.11.2019 to 31.12.2019. Further, I restricted the training and validation set by excluding bins for which no volume was traded in the following bin or lagged values are not available.

I cross-validated the respective parameter space by first training the model on the training set, and then evaluating it on the chronologically following validation set for each parameter combination. After obtaining an accuracy evaluation of each combination, the model gets fitted with the hyperparameters of the best validation performance on the combined training and valdation set.

Further, I also give a brief description of the models, and how I used them for the trading experiments. Before doing so, I introduce the mathematical notation. Let $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ denote the training data set containing n observations on which the respective model gets fitted whereas $x_i \in \mathbb{R}^K$ denotes the feature vector with K features and y_i the target class variable for the i-th observations respectively. In addition, the Decision Tree is denoted [4] as a function $T_D^q(x)$ with D denoting the set which it was trained on, q the selection of features it was trained on and x feature vector as an the function argument.

4.3.1 Logistic Regression

The Logistic Regression model analyzes the relationship between multiple independent variables or features and a categorical dependent variable or target, and estimates the probability of occurrence of an event by fitting data to a logistic curve. There are two models of logistic regression, binary Logistic Regression and multinomial Logistic Regression. Binary Logistic Regression is used in this application, since the dependent variable is dichotomous (either *down*- or *up*-movement, i.e., 0 or 1), whereas the multinomial one would be used to classify instances to multiple classes.

The building block for the Logistic Regression is the multiple regression model ([23], [36]),

$$y = \sum_{j=1}^{J} \beta_j X_j + \epsilon = X^T \beta + \epsilon \tag{4}$$

with $y \in \mathbb{R}$ denoting the binary target, β_j the parameter assigned to the j-th feature X_j and ϵ the error term. A problem that occurs when applying the multiple regression model is that the predicted values might not fall between 0 and 1, thus not being eligible for estimating a probability $p(X)$. When fitting a straight line to approximate a binary response coded as 0 or 1, it can occur that for some values of X $p(X) < 0$ and $1 < p(X)$ for others (unless the range of X is limited) [23]. Therefore, to avoid this problem in the Logistic Regression, the *logistic function* gets wrapped around the multiple regression model,

$$p(X,\beta) = \frac{\exp(X^T\beta)}{1+\exp(X^T\beta)} = \frac{1}{1+\exp(-X^T\beta)} \tag{5}$$

thus, the model can be written as,

$$y = p(X,\beta) + \epsilon \tag{6}$$

In order to estimate the coefficients β, the *maximum likelihood* method can be used. The objective is to find estimates for β such that the predicted probability $p(x_i,\beta)$ of the price movement for each observed feature vector x_i, using equation 5, corresponds as closely as possible to the observed price movement y_i. This objective can be formalized using a mathematical equation called a *likelihood function* [23]:

$$\mathscr{L}(\beta) = \prod_{i=1}^{n} p(x_i,\beta)^{y_i}(1-p(x_i,\beta))^{1-y_i} \tag{7}$$

Therefore, the optimization problem for the Logistic Regression takes the following form:

$$\beta^* = \underset{\beta\in\mathbb{R}^J}{\arg\max}\, \mathscr{L}(\beta) \tag{8}$$

I decided to use the Logistic Regression for predicting price movements, because it is relatively easy to interpret and it was used extensively as a benchmark in the related literatur ([12], [11]). For this paper, I relied on the Scikit-learn implementation of [1] for the Logistic Regression and follow the parameters outlined in [12].

It seems reasonable to suppose that not only a few returns and volumes but most of them exhibit considerable explanatory effect. Therefore, I chose the L2- over L1-regularization for the estimated parameters, which was also done in the related literatur ([12], [11]). The optimal L2-regularization was determined among 100 values on a logarithmic scale from 0.0001 to 10,000 via cross-validation as explained above on the respective training set. Further, I restricted the maximum number of iterations to 150.

4.3.2 Random Forest

Before outlining the algorithm for the Random Forest, I explain the essential building block of the Random Forest which is the Decision Tree.

The Decision Tree is a non-parametric machine learning method used for classification and regression. It predicts the response with a set of if-then-else decision rules derived from the data. The deeper the tree, the more complex the decision rules and the closer the model fits the data. The Decision Tree builds classification or regression models in form of a tree structure. Each node in the tree further partions the feature space into smaller and smaller subsets, while at the same time an associated Decision Tree is incrementally developed. The final result is a tree with decision nodes and terminal nodes. A decision node has two or more branches. Leaf nodes represent the actual classification or final decision. The topmost decision node in a tree which corresponds to the best predictor is called the root node. Decision Trees can handle both categorical and numerical data [23].

An example of such a tree is depicted below in figure 3.

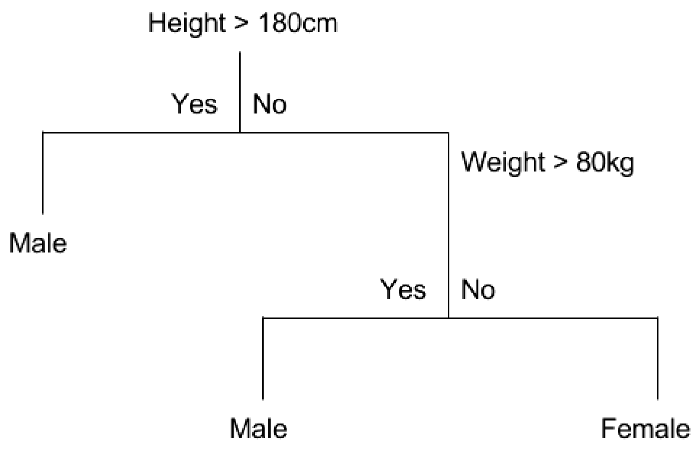


Figure 3: Given a data set with two features height and weight, and gender as the target variable, this example tree stratifies the two-dimensional feature space into three distinct subset each represented by the terminal nodes at the bottom. The stratification occurs at the two deciding nodes depending either on whether its height is above 180 cm and its weight is above 80kg.

In the following, I describe the CART algorithm for tree building as specified in [4]. The basic idea of tree growing is to choose a split among all the possible splits at each node such that the resulting child nodes are the "purest". In this algorithm, only univariate splits are considered. That is, each split depends on the value of only one predictor variable. All possible splits consist of possible splits of each predictor.

A tree is grown starting from the root node by repeatedly using the following steps on each node in the following algorithm:

Algorithm 1: Binary Splitting [4]

```
Data: Training set $D$ with $J$ features
Result: Decision Tree $T_D^{q=J}$
Initialize $DecisionTree$ (i.e. $T_D^{q=J}$)
while $DecisionTree.checkStoppingCriterion()$ do
    /* Loop over nodes associated with decision rules of $T_D^{q=J}$ */
    for $t \in DecisionTree.nodes$ do
        (i) Find best split $s$ for each feature $X_j$: For each feature $X_j$, there exist $J-1$
            potiential splits, whereas $J$ is the number of different values for the respective
            feature. Evaluate each value $X_{i,j}$ at the current node $t$ as a candidate split point
            (for $x \in X_j$, if $x \leq X_{k,i} = s$, then $x$ goes to left child node $t_L$ else to right child
            node $t_R$). The best split point is the one that maximize the splitting criterion
            $\Delta i(t, t_L, t_R)$ the most when the node is split according to it.
            $j^*, s^* \leftarrow \underset{j \in \{1,\dots,J\}, s_j \in \{x_j : x_j \in t\}}{\arg\max} \Delta i(t, t.split(s_j, j))$
        (ii) Add decision rule associated with split $s^*$, feature $x_{j^*}$ and node $t$ to
            decision tree $T_D^{q=J}$
            $DecisionTree.splitNode(t, s^*, j^*)$
```

The splitting criterion $\Delta i(t)$ is defined as,

$$\Delta i(t, t_L, t_R) = i(t) - p_L i(t_L) - p_R i(t_R), \tag{9}$$

with the Gini Coefficient being used as the impurity measure $i(t)$ in the CART algorithm,

$$i(t) = \sum_{c \in C} p(c|t)(1 - p(c|t)) \tag{10}$$

However, there exist variants of the algorithm which incroporate different impurity measures such as Information Gain or Variance Reduction [40].

Since the Decision Tree tends to overfit the data ([25], [18]), especially in the case of large Decision Trees, the Random Forest was introduced to mitigate this problem. This is done by casting a vote based on an ensemble of individual Decision Trees trained on bootstrapped samples from data D. Further, in order to decorrelate the Decision Trees from each other, only a random subset of features is considered at step (i) of algorithm 1 each time a node is split further, i.e., an additional decision rule gets added. This process is further illustrated in algorithm 2 below.

Algorithm 2: Generation of Decision Tree ensemble for the Random Forest 2

```
Data: Training set $D$, ensemble size $n_E$
Result: Tree ensemble $E = \{T^q_{D_1}, ..., T^q_{D_{n_E}}\}$
/* Generate $n_E$ Decision Trees                                   */
for $i \in \{1, ..., n_E\}$ do
    /* Draw bootstrapped sample $D_i$ from $D$                      */
    $D_i \leftarrow bootstrap(D)$
    /* Grow Decision Tree as specified in algorithm 1, except only
       use a randomly selected feature subset of size $q$ in step $(i)$
       */
    $tree \leftarrow DecisionTree.fit(D_i, features\_selection = q)$
    $E \leftarrow E \cup tree$
```

The reason that the Random Forest might be a suitable candidate for predicting price movements lies in its ability to model complex non-linear decision boundaries and to grow arbitrarily large models. In addition, Random Forests in this configuration are the best method in similar works as in [26] and the method of choice machine learning application on monthly stock market data in [31]. Thus, Random Forests can be seen as a viable benchmark for experimental setups such as this one.

In this application of the Random Forest, I conducted a cross-validated grid-search over the number of Decision Trees in the ensemble $n_E \in \{100, 500, 1000\}$ and the maximum depth of each tree $d_{T^q_{D_i}} \in \{3, 5, 10, 15\}$ taking advice from [26], [11] and [1].

4.3.3 AdaBoost

The Adaptive Boosting (AdaBoost) is a classification machine learning algorithm that combines multiple so called weak learners. First, it fits weak classifiers, a Decision Tree in this application, on the original dataset. Then, it proceeds to fit additional weak classification models on the dataset for which weights of previously incorrectly classified instances are adjusted such that the following classifiers focus more on difficult cases [13]. According to [47], under zero-loss, the missclassification rate is given by $1 - \sum_{k=1}^{K} E_x \mathbb{1}_{C(X)=k} P(C = k|X)$. Therefore, the *Bayes Classifier*

$$C^*(x) = \arg\max_k P(C = k|X = x) \tag{11}$$

will minimizes this quantity with the missclassification error rate, i.e., *Bayes Error Rate*, equal to $1 - E_X max_k P(C = k|X)$. The objective of the AdaBoost algorithm is to iteratively approximate the Bayes Classifier $C^*(x)$ by combining *weak learners* [47]. These are classification models wich have only a small predictive power. In this application, I use small Decision Trees with a depth of up to three. First, the AdaBoost fits a classifier with the unweighted training sample that assigns class labels to each instance in the training data set. If the respective assigned label does not equal the true one, the weight of that training data instance is increased. Then, the AdaBoost proceeds to train a second classifier including the new weights. For instances that got misclassified, the weights get boosted, and the procedure gets repeated.

The AdaBoost algorithm is illustrated below:

Algorithm 3: AdaBoost [47], [13]

```
Data: Training set $D$ with $K$ features
Result: $AdaBoost(x)$
Initialize weights $w_i = 1/n$ for $i = 1, ..., n$
Initialize DecisionTree
for $m \in \{1, ..., M\}$ do
    /* Apply weights to the data set                                   */
    $\widetilde{D} = \text{adjustDataSet}(D, \{w_i\}_{i=1}^{n})$
    /* Fit m-th classifier                                             */
    $T_{\widetilde{D}}^{(m)} \leftarrow \text{DecisionTree.fit}(\widetilde{D})$
    /* Compute weighted error rate                                     */
    $err^m \leftarrow \sum_{i=1}^{n} w_i \cdot \mathbb{1}(c_i \neq T_{\widetilde{D}}^{(m)}(x_i)) / \sum_{i=1}^{n} w_i$
    /* Compute $\alpha$ parameter                                      */
    $\alpha^{(m)} \leftarrow ln\frac{1-err^m}{err^m}$
    /* Readjust weights                                                */
    $w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{1}(c_i \neq T_{\widetilde{D}}^{(m)}(x_i))\right), \; for \; i = 1, ..., n$
    /* Renormalize weights                                             */
    $\{w_i\}_{i=1}^{n} \leftarrow \text{renormalize}(\ \{w_i\}_{i=1}^{n}\ )$
return $C(x) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{1}(T_{\widetilde{D}}^{(m)}(x_i) = k)$
```

According to [15], the AdaBoost is relatively fast, and has a low amount of parameters to tune. It requires no prior knowledge about the weak learner and so can be flexibly combined with any method for finding weak hypotheses. Further, the AdaBoost is supposed to be a good candidate for to identifying outliers, because it focuses its weight on the hardest examples, the examples with the highest weight often turn out to be outliers ([14], [15]). This property could be highly valuable when trying to predict price movements, since these outliers could exhibit the most reliable probability signals for the trading algorithm. On the other hand, boosting can fail to perform well given insufficient data and seems to be especially susceptible to noise ([15], [6]). Therefore, this experiment shows if and how this tradeoff will be resolved. In this application of the AdaBoost, I conducted a cross-validated grid-search over the number of weak learners, i.e., Decision Trees, $M \in \{500, 1000\}$, the maximum depth of each tree $d_{T^{(m)}} \in \{1, 3\}$, and the learning rate $\lambda \in \{0.001, 0.01, 0.1\}$ taking advice from [26], [19] and [16].

4.3.4 Deep Neural Network

In this chapter, I briefly describe the Deep Neural Network (DNN) according to [7] and [26]. A deep neural network consists of an input layer, one or more hidden layers, and an output layer. The topology of the network can be seen in figure 4.

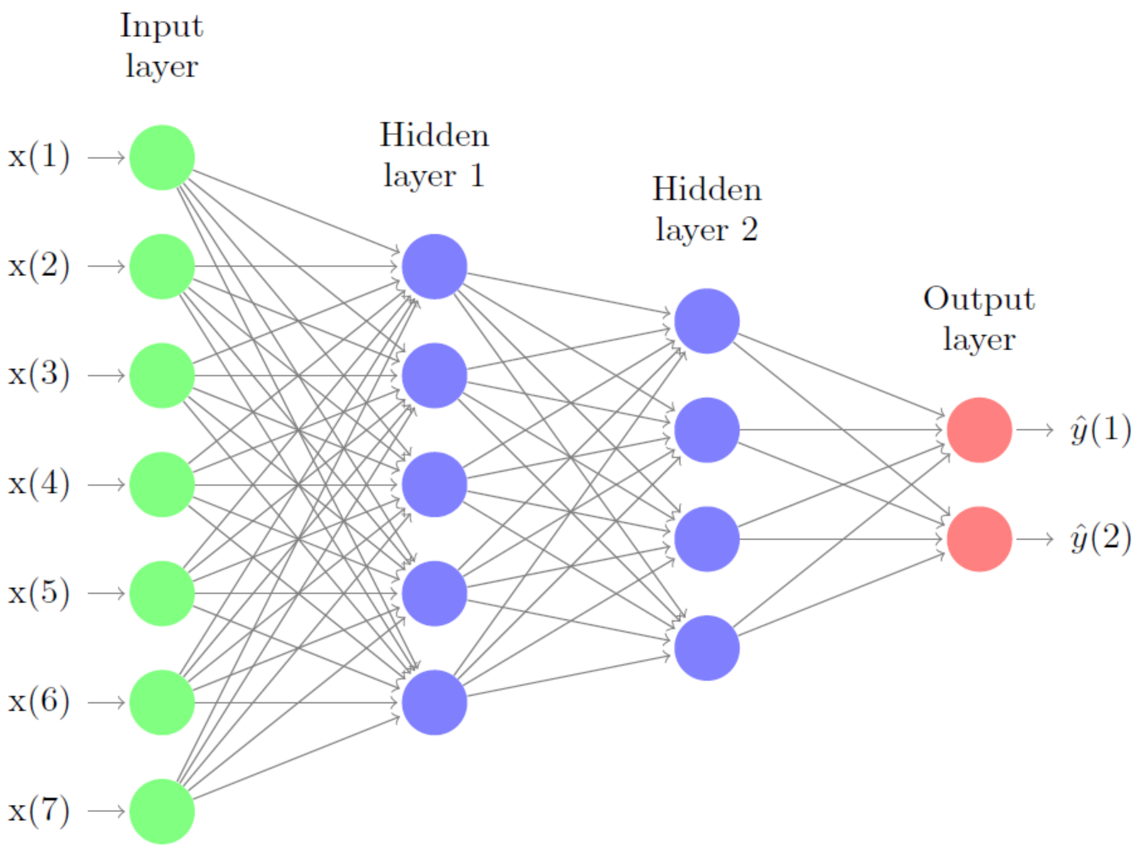


Figure 4: This figure illustrates the topology of a deep neural network consisting of two hidden layers, seven features and two output states [7].

The input layer matches the feature space in dimensions, thus there are as many input neurons as predictors. The output layer is either a classification or regression layer to match the output space. All layers consist of neurons, which are themselve connected to the ones from the neighboring layers. Thus, each neuron in layer l is connected to all neurons in the following layer $l+1$. Further, each neuron in a input and hidden layer has a bias parameter b_j^l similar to actual biological neurons. Therefore, each neuron j of layer l receives a weighted combination α_j^l of the n_{l-1} outputs of the neurons $i = 1, ..., n_{l-1}$ in the previous layer $l-1$ as input,

$$\alpha_j^l = \sum_{i=1}^{n_{l-1}} w_{i,j}^l x_i^{l-1} + b_j^l, \tag{12}$$

with $w_{i,j}^l$ denoting the corresponding weight in layer l for the input α_j^l of the previous layer $l-1$. b_j^l denotes the bias for neuron j of layer l. The weighted combination α of equation 12 is transformed by a so-called activation function f, such that the output signal $f(\alpha)$ is forwarded to the neurons in layer $l+1$. Comparing equation 12 and the Logistic Regression from chapter 4.3.1, the similarities become apparent. In that sense, a DNN can be interpretet as multiple sequences of Logistic Regression chained together with different activation functions converging into the final model output. The activation function used in this application is ReLU (rectified linear activation unit) ([2], [3]) $f : \mathbb{R} \to \mathbb{R}$,

$$f(\alpha^l) = max(\alpha^l, 0) = (max\{0, \alpha_1^l\}, ..., max\{0, \alpha_{n_l}^l\})^T \tag{13}$$

The choice for this activation function is inspired by the sudden burst of action potential releases of biological neurons [45].

Let $W = \bigcup_{l=1}^{L-1} W_l$, with $W_l = (w_{i,j}^l) \in \mathbb{R}^{n_{l-1} \times n_l}$ denote the weight matrix that parametrizes layers l and $l+1$ for a network consisting of L layers. Further, let the set $B = \bigcup_{l=1}^{L-1} b_l$, with $b_l = (b_j^l) \in \mathbb{R}^{n_{l-1}}$ denote the vector of biases for layer l such that the objective is to find the optimal $\theta = (W, B)$ parameters. The DNN is optimized by adapting weights θ in order to minimize the error on the training data. This is done by minimizing the loss function $\mathscr{L}(\theta)$ for each training example $(x_i, y_i) \in D$. Since this experimental setup consists of a classification problem, the loss function is the cross-entropy,

$$\mathscr{L}(\theta) = -\sum_{i=1}^{n} \sum_{k=1}^{K} ln(m(y = y_k | x_i; \theta)) \cdot y_k, \tag{14}$$

with $m(y = y|x_i)$ denoting the DNN fully parametrized by θ as a function of an observed feature vector x_i. Since the softmax function [7] is used to transform the output from the last hidden layer, the DNN can expressed as follows:

$$m(y = y_k | x_i) = \frac{\alpha_c^L}{\sum_{j=1}^{K} \alpha_c^L} \tag{15}$$

The loss function 14 is minimized by stochastic gradient descent, with the gradient of the loss function $\Delta_\theta \mathscr{L}(\theta)$ being calculated via backpropagation [26].

$$f^{'}(\alpha^l) = (\mathbb{I}(\alpha_1^l > 0), ..., \mathbb{I}(\alpha_{n_l}^l > 0))^T \tag{16}$$

Following [38], [39], and [7], the backpropagation learning algorithm based on the method of stochastic gradient descent (SGD) updates the parameters θ after randomly drawing an observation i,

$$\theta = \theta - \lambda \nabla_\theta \mathscr{L}(\theta, i) \tag{17}$$

with λ denoting the learning rate. Algorithm 4 below provides a high level pseudo-algorithmic description of the sequential version of SGD according to [7].

Algorithm 4: Stochastic Gradient Descent [7]

```
Data: Training set $D$ features and parameter space of DNN
Result: Optimized parameter values $\theta$ for DNN
/* Draw initial parameter values from normal distribution */
$\theta \leftarrow r$ with $r = (r_i)_{i=1}^n$ and $r_i \in \mathscr{N}(\mu, \sigma), \forall i$
/* Get loss according to initial parametrisation */
$\mathscr{L} \leftarrow 0$
for $i \in \{1, ..., n\}$ do
    $\mathscr{L} \leftarrow \mathscr{L} + \mathscr{L}_i(\theta)$
/* As long as loss above threshold $\tau$, continue to update $\theta$ */
while $\mathscr{L} \geq \tau$ do
    for $i \in \{1, ..., n\}$ do
        /* Draw random sample from data $D$ */
        $i \leftarrow drawRamdomly(D)$
        $\theta \leftarrow \theta - \lambda \nabla_\theta \mathscr{L}(\theta, i)$
    $\mathscr{L} \leftarrow 0$
    for $i \in \{1, ..., n\}$ do
        $\mathscr{L} \leftarrow \mathscr{L} + \mathscr{L}_i(\theta)$
```

In order to describe the network's topology, the following notation gets used: I-H1-H2-H3-H4-O. I denotes the number of input neurons, H1, H2, H3 and H4 the number of hidden neurons in hidden layers 1, 2, 3, and O the number of output neurons. According to [26], a popular rule, which I also abide by, is to set the number of neurons in the first hidden layer H1 of a DNN to the number of inputs. To introduce a closer fit to the data, I also set H1 = H2. A bottleneck gets introduced by setting H3 = 15 and H4 = 10, thus enforcing a reduction in dimensionality in line with [43] and [7]. Let K be the number of features of the data. Then, the topology specification results in I-H1-H2-H3-O = K-K-K-15-10-2 with input layer I matching the number of features. The model got trained across 400 epochs, i.e., the data gets passed 400 times over the training set, as in [21].

As a disclaimer, I would like to point out that according to [46], the design of a DNN is more of an art than a science. Therefore, I mainly followed best practices and intuitions to calibrate the DNN. However, for the purpose of this paper, it can be argued that it is sufficient, since it takes the operational costs aspect of such an endeavour into account. The actual reason for using the DNN is its ability to model complex decision boundaries [24]. This flexibility could be of great use for predicting price movements.

4.4 Trading Algorithm

For the trading phase, I proceeded similarly to [11], [17] and [26]. After having trained each model for different specifications, probability estimates got generated for each class based on the trading sets feature's. These probabilities for each bin were then used as signals in bin t for each coin in order for the algorithm to decide whether to enter or close a position invested in a coin in the next bin in $t + 1$. In this work, I refer to these positions as active. More specifically, the algorithm compares the probability estimate for each coin and decides whether to close the current position and enter a new one. The coin with the highest probability estimate for a down movement is considered as a candidate for the short position if its probability is also above a certain threshold, since it is most likely to go down. The algorithm proceeds analogously for the long position. If a coin gets chosen this way and the position for this movement is active, then it proceeds to close this position. Therefore, a long and a short position gets opened at most per bin t. In order get a better estimate of the return, I entered 60 initial short and long positions at different points in time, thus the resulting portfolio has 120 active positions at most simultaneously. Then, I proceeded to calculate aggregate values for each of these positions. To render the backtest more realistic, I incorporated the following constraints:

Minimum duration: Any active position has a minimum duration of $d = 120, 240$ before considering closing it.

Execution gap: To account for the time it takes to generate a probability signal and submit the order accordingly, I introduce an execution gap of one minute. This means that when generating the signal from a bin in t, the order gets executed in $t + 1$ the earliest.

Minimum volume: Orders for opening or closing a position only get executed in bin t if any volume was traded in the respective bin.

Order cancel: If after submitting the order in t the traded volume in bin $t+1$ is zero, the order gets canceled, and a new probability signal gets generated in bin $t + 1$.

Transaction cost: For every order execution a transaction cost of ϵ bps gets subtracted. Thus, the opening and closing of a position costs $2 \times \epsilon$ bps.

Keep active position: If the probability signal yields the same coin as the one from the current active position, then keep the same position open for another $d = 120, 240$ minutes before again generating another probability signal.

The algorithm described above can also be expressed in a pseudo-algorithmic way:

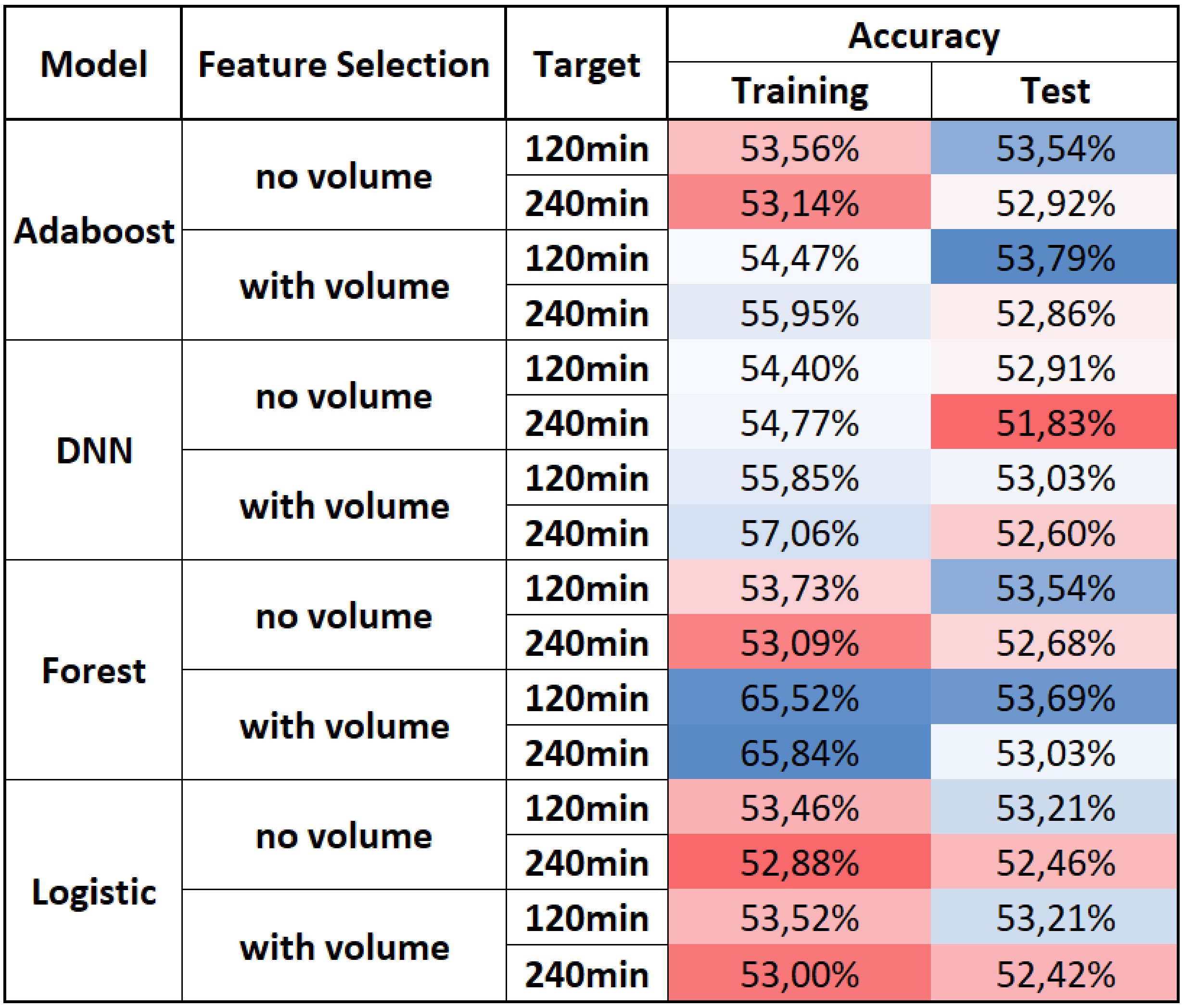
Algorithm 5: Pseudo-Algorithm for a single position (short and long are analogous)

```
Data: Minute-binned Test-OHLC-Data $OHLC$ and its estimated probabilities
Result: Trading decisions and its realized returns
Initiate $Position$
$row\_number \leftarrow 0$
while $row\_number < (max\_row\_number - delta)$ do
    /* Load row data given by $row\_number$ */
    $row \leftarrow OHLC.getRow(row\_number)$
    /* Obtain maximum probability pair $max\_prob\_pair$ */
    $max\_prob\_pair \leftarrow row.getMaxProbPair()$
    /* Determine whether probability condition is fulfilled */
    if $row.getProb(\ max\_prob\_pair\ ) > threshold$ then
        if $Position.active == False$ then
            Position.open()
        else if $max\_prob\_pair\ != Position.pair$ then
            Position.close()
    else
        if $Position.active == True$ then
            Position.close()
    /* Skip minute bins based on current position */
    if $Position.active == True$ then
        $row\_number \leftarrow row\_number + delta$
    else
        $row\_number \leftarrow row\_number + 1$
```

5 Results

5.1 General Results

After training the models, I evaluated their accuracies on the remaining trading, i.e., test data ranging from 01.11.2019 to 31.12.2019. Before applying the models on the test data, bins for which no volume was traded got excluded, as it was already done for the training data. The accuracy for each model specification are illustrated below in figure 1.



Model	Feature Selection	Target	Accuracy	
			Training	Test
Adaboost	no volume	120min	53,56%	53,54%
		240min	53,14%	52,92%
	with volume	120min	54,47%	53,79%
		240min	55,95%	52,86%
DNN	no volume	120min	54,40%	52,91%
		240min	54,77%	51,83%
	with volume	120min	55,85%	53,03%
		240min	57,06%	52,60%
Forest	no volume	120min	53,73%	53,54%
		240min	53,09%	52,68%
	with volume	120min	65,52%	53,69%
		240min	65,84%	53,03%
Logistic	no volume	120min	53,46%	53,21%
		240min	52,88%	52,46%
	with volume	120min	53,52%	53,21%
		240min	53,00%	52,42%

Figure 5: This table illustrates the accuracies for the different models being used on training and test set for different combinations of features and targets. Red colors indicate relatively low and blue ones relatively high accuracies compared to the other respective training and test results.

As one can see above, the accuracies are generaly fairly low for both training and test set, which is not out of the ordinary when comparing the model performances to the results of other methods (see [30]). This indicates that the data contains a high amount of noise and low explanatory power, which supports the efficient market hypothesis. Even when including volume, the test accuracy remains low.

For the training set, the inclusion of volume increases the training accuracy when comparing the respective counterpart without volume. This ought to be expected, since there are more features to fit the data with. Methods such as DNN and Random Forest, which tend to fit the noise more closely, receive the greatest increase in training accuracy. However, these increases in training accuracies do not translate into increases in test accuracy, which remains almost unchanged.

Changing the duration of an active position, i.e., increasing the lead d from 120 to 240 min for target $Y^c_{t+d,t}$ decreases test accuracies consistently. This can be explained by the fact that price movements over longer periods of time exhibit greater variance, thus leading to more uncertain predictions regarding price movements. Since the accuracies for each model and specification are rather low in figure 5, one should consider increasing the threshold. Otherwise the postive returns from the correct trading decision signal get canceled out almost completely by the wrong ones. Further, after introducing transaction costs for each trade, false trades can cause the overall return to drop considerably if conducted too often which can be seen in chapter 5.2.

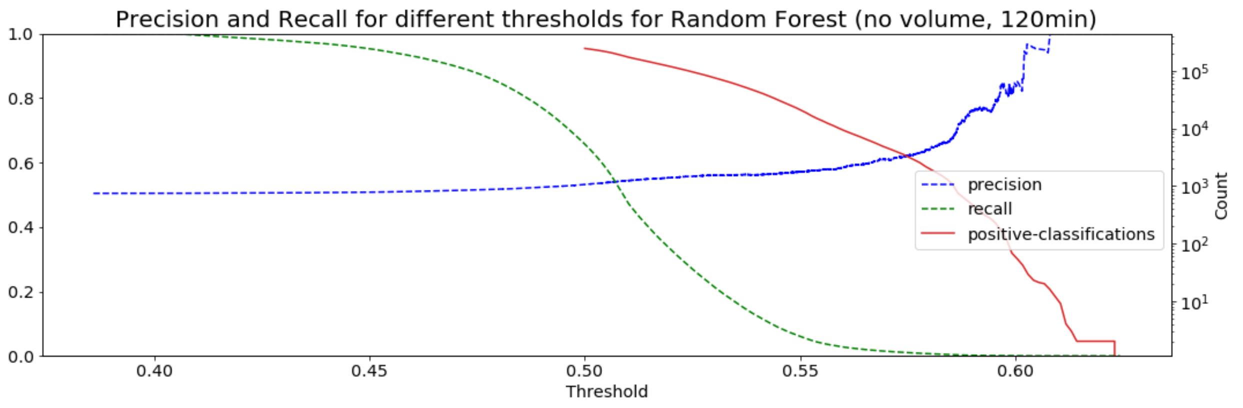
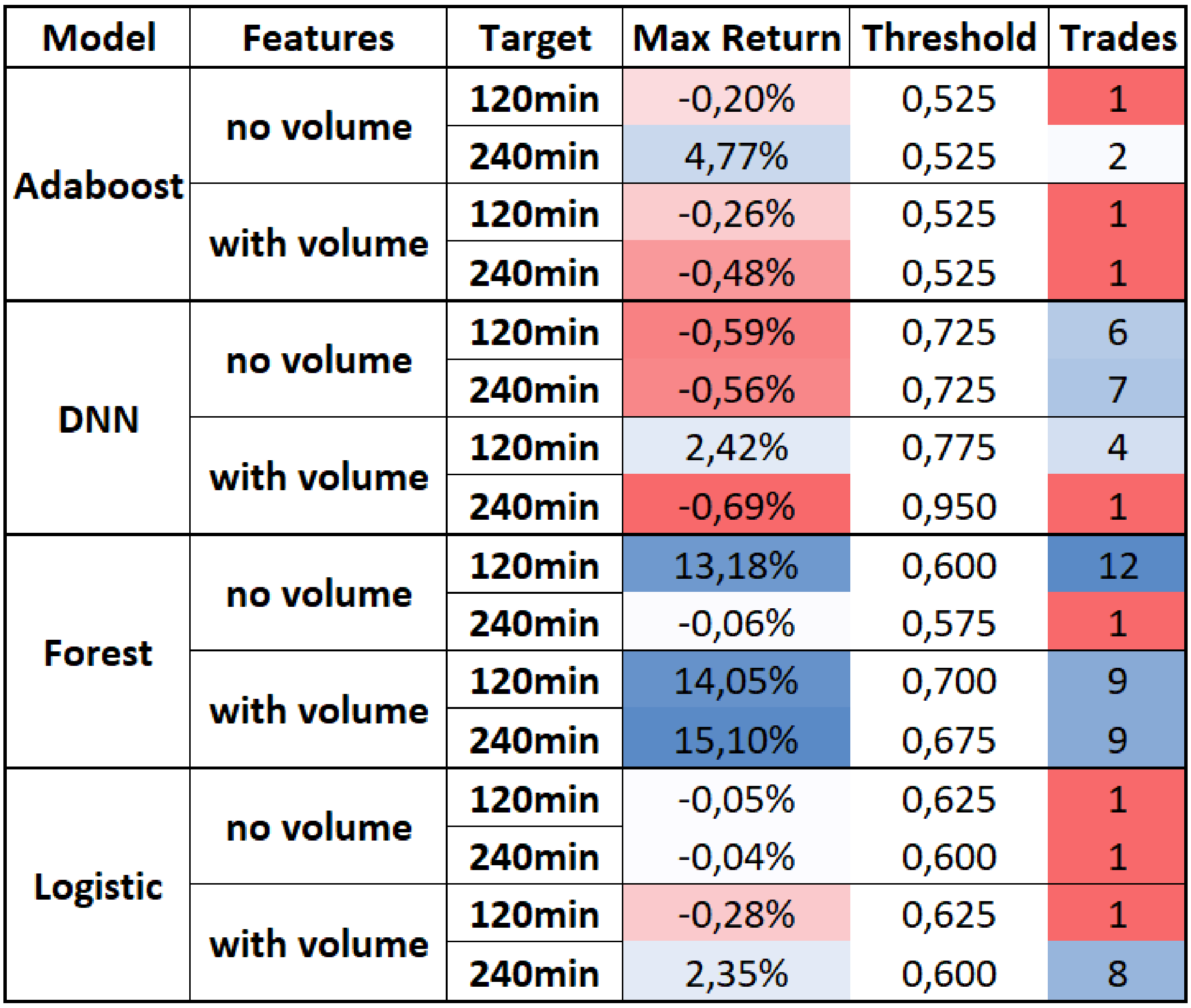


Figure 6: This figure illustrates the precision, recall and up-classifications of the Random Forest for different thresholds without taking volume into consideration.

Figure 6 illustrates the tradeoff between precision and positive-classification for increasing tresholds. When conducting trading with transaction cost, one should aim for increasing the precision, since false-positive decrease returns considerably. However, by increasing the threshold in order to improve the precision, less positive-classifications, i.e., long positions are conducted. Thus, it is imperative to find a balance between precision and number of classifications. In chapter 5.2, I demonstrate how changing thresholds leads to drastic differences in terms of overall return.

5.2 Strategy Performance

In this chapter, I outline the performance after applying the trading algorithm on the trading set as described in chapter 4.4. Figure 7 below illustrates total maximum returns with 15bps transaction cost and its respective probability threshold for each model and specification.



Model	Features	Target	Max Return	Threshold	Trades
Adaboost	no volume	120min	-0,20%	0,525	1
		240min	4,77%	0,525	2
	with volume	120min	-0,26%	0,525	1
		240min	-0,48%	0,525	1
DNN	no volume	120min	-0,59%	0,725	6
		240min	-0,56%	0,725	7
	with volume	120min	2,42%	0,775	4
		240min	-0,69%	0,950	1
Forest	no volume	120min	13,18%	0,600	12
		240min	-0,06%	0,575	1
	with volume	120min	14,05%	0,700	9
		240min	15,10%	0,675	9
Logistic	no volume	120min	-0,05%	0,625	1
		240min	-0,04%	0,600	1
	with volume	120min	-0,28%	0,625	1
		240min	2,35%	0,600	8

Figure 7: This figure illustrates total maximum returns with 15bps transaction cost, its respective probability threshold for each model and specification and total trades conducted.

Except for the random forest, which outperforms any other method significantly, most methods struggle to achieve positive returns. AdaBoost, DNN, and Logistic Regression only generate negative returns, except for one feature and target specification combination. Interestingly, this combination is different for each model, indicating no consistant explanatory power across models. Compared to the 42% short return of the equal-weight portfolio consisting of an equal investment in all ten coins and holding throughout the trading period, the models are not able to beat simple diversified holding strategies.

Increasing the forecast duration from 120 to 240 min does not impact returns consistently with a correlation to returns of roughly -0,08. This indicates that the potential upside of higher price changes and less frequent transaction costs are mitigated by the increased difficulty for the model to predict the price movement. Also including volume has no consistent impact on returns.

The thresholds which maximize returns remain stable for AdaBoost and Logistic Regression across different specifications. This is not the case for Random Forest and DNN for which this threshold is higher, which is indicative of a closer fit. Further, the probability mass for the AdaBoost seems to be close to 0.5, since for a threshold of only 0,525 the number of conducted trades is either 1 or 2. This finding is also in line with the literature which states that decision trees and boosting methods produce distorted class probability distributions (see [32] and [44]). Figure 8 illustrates precision, recall and up-classifications of the AdaBoost for different thresholds without taking volume into consideration. One can see that, only slight increases in thresholds lead to drastic changes in precision and positive classifications. The Random Forest on the other hand exhibits a rather small changes for increasing thresholds in figure 6.

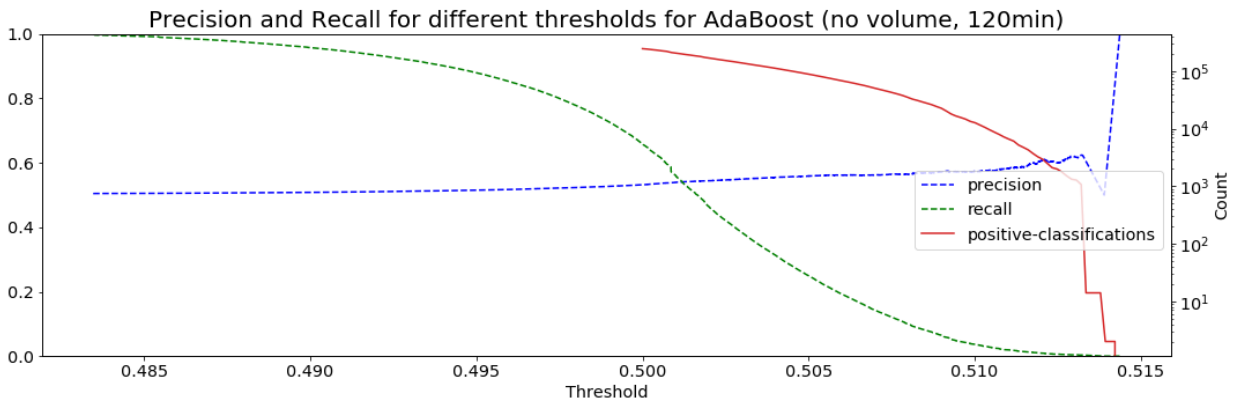


Figure 8: This figure illustrates precision, recall and up-classifications of the AdaBoost for different thresholds with a prediction duration of 120 min and only return features.

DNN and Random Forest do not have these biases and predict less distorted probabilities (see appendix figure 19), which is also supported by the literature [32]. Overall, when calibrating the threshold to maximize returns, only a small amount of trades are conducted for each model and specification. This can be attributed to the aforementioned tradeoff between precision and positive-classifications when increasing the threshold, which is consistent across all specifications (see appendix figure 15 to 18). However, only considering maximum returns, an interesting relationship emerges. The greater the number of conducted trades, the greater the maximum returns (correlation 0,77), due to a higher amounts of profitable trades. This is insofar surprising as a higher amount of trades due to lower thresholds consistently leads to less return as can be seen in figure 9 below.

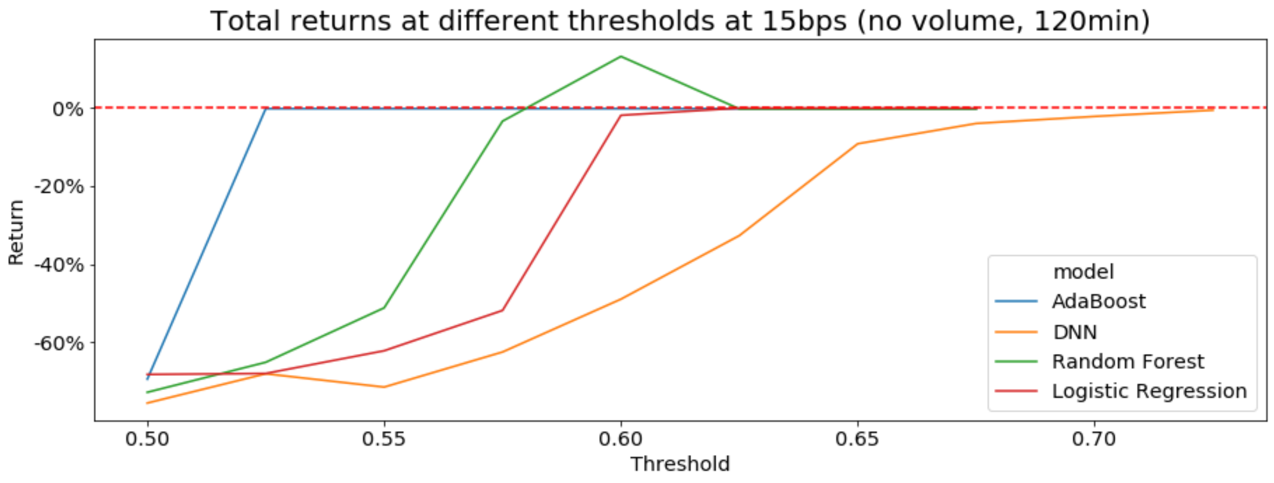


Figure 9: This figure illustrates total returns at a transaction cost of 15 bps over the trading period for different thresholds. The four models were trained on features only containing returns and a forecast duration of 120 min.

All models start off with similar negative returns and converge to zero as the threshold progresses. This can be explained by the tradeoff illustrated in figure 8 and 6 as the precision and accuracy is very low at thresholds near 0,5. The convergence is due to the small amount of trades occuring at higher thresholds, thus no trades are being conducted eventually. This finding is consistent across models and specifications (see appendix figure 19).

5.3 Single-Coin Models

In this chapter, findings of chapter 5.2 above get contrasted with the performance of the models trained only on BTC. The motivation for doing is due to the objection that the models would perform better if only one coin would be used, since this could reduce the noise in the data. We restricted the sample to only BTC, because the data of the other coins exhibits a considerable amonunt of missing bins. Further, the target got changed to a simple dummy for the sign of the respective future return, because the returns of only one coin do not imply how it performs compared to the others in terms of median values. Thus, the signals from this data set are even less sufficient for forecasting the original target.

Figure 10 below illustrates the performances of the models only trained on BTC data with 15bps transaction cost and with forecast duration of 120 min.

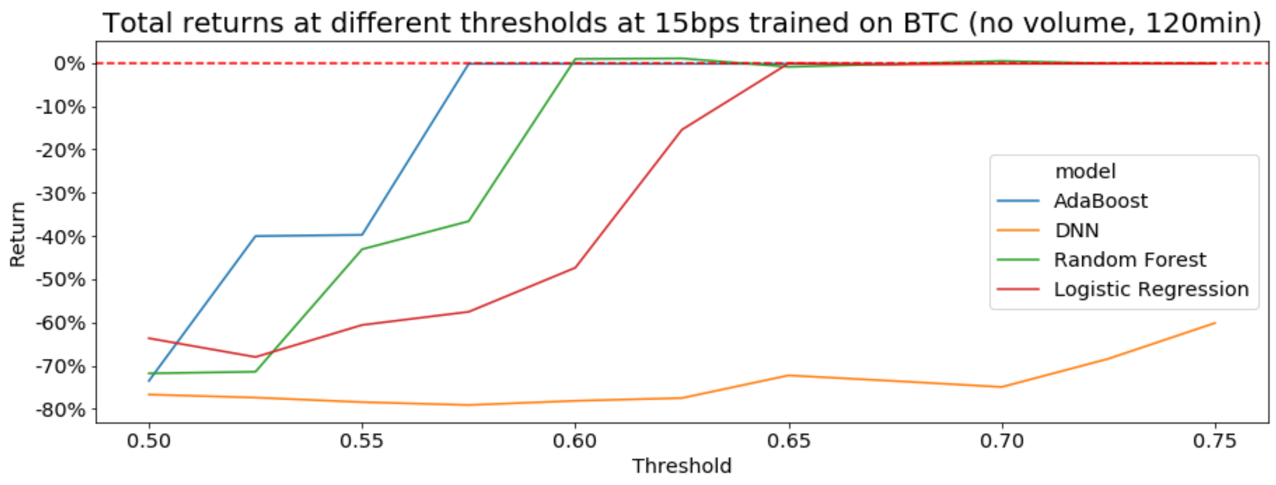


Figure 10: This figure illustrates total returns at a transaction cost of 15 bps over the trading period for different thresholds. The four models were trained on features only containing returns and a forecast duration of 120 min from BTC data.

As one can see above, most models struggle to reach positive returns. This is most pronounced for the DNN which remains consistently at very low returns. Further, changing target and feature selection does not impact the development of returns significantly (see appendix 22).

5.4 Further Analyses

In this chapter, I analyse how the different return deltas impacted the predictions of the model.

Figure 11 illustrates the feature importances for the Random Forest according to MDI [28] for different return deltas in minutes (see chapter 4.2). For this particular graph, the Random Forest was trained on features containing only returns and a forecast duration of 120 min. The overall shape resembles a concave graph peaking at around 80 min and then decreasing in importance.

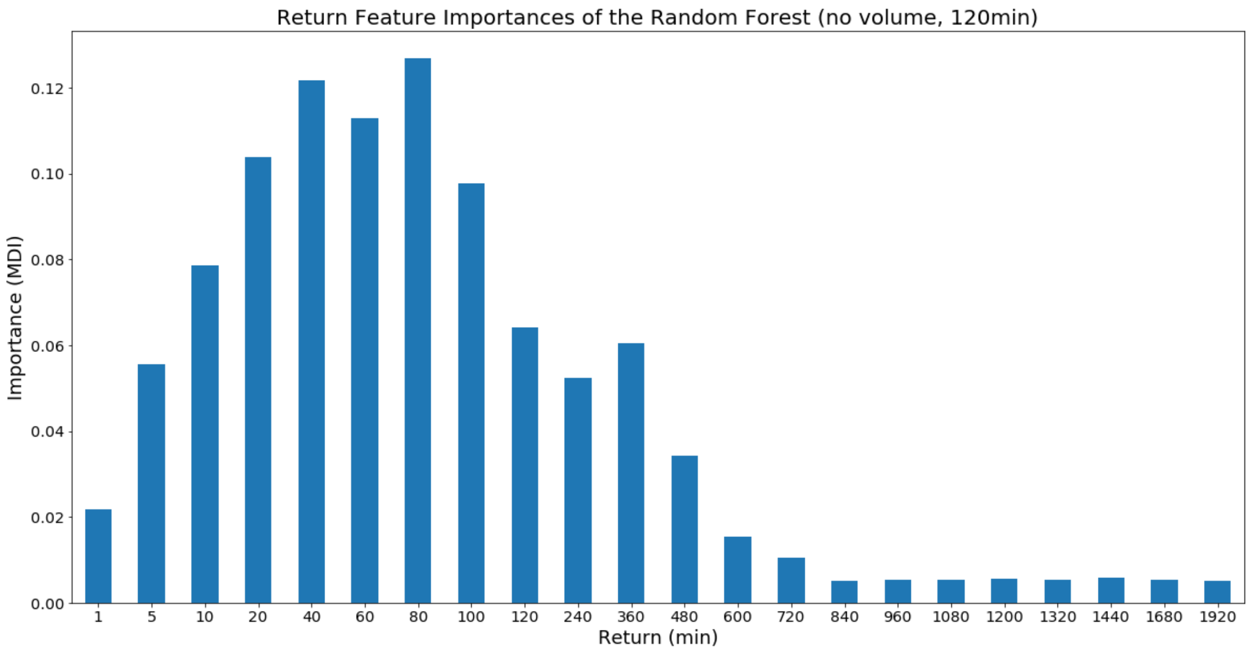


Figure 11: This figure illustrates the feature importances of the Random Forest according to MDI [28] for different return deltas in minutes as described in chapter 4.1. The Random Forest was trained on features containing only returns and a forecast duration of 120min.

As one can see, after a return delta of 10 hours, the relative importance remains low. This indicates that returns over a long period of time exhibit a relatively low explanatory power for predicting price movements, while those near the middle of the forecast duration have the highest explanatory power.

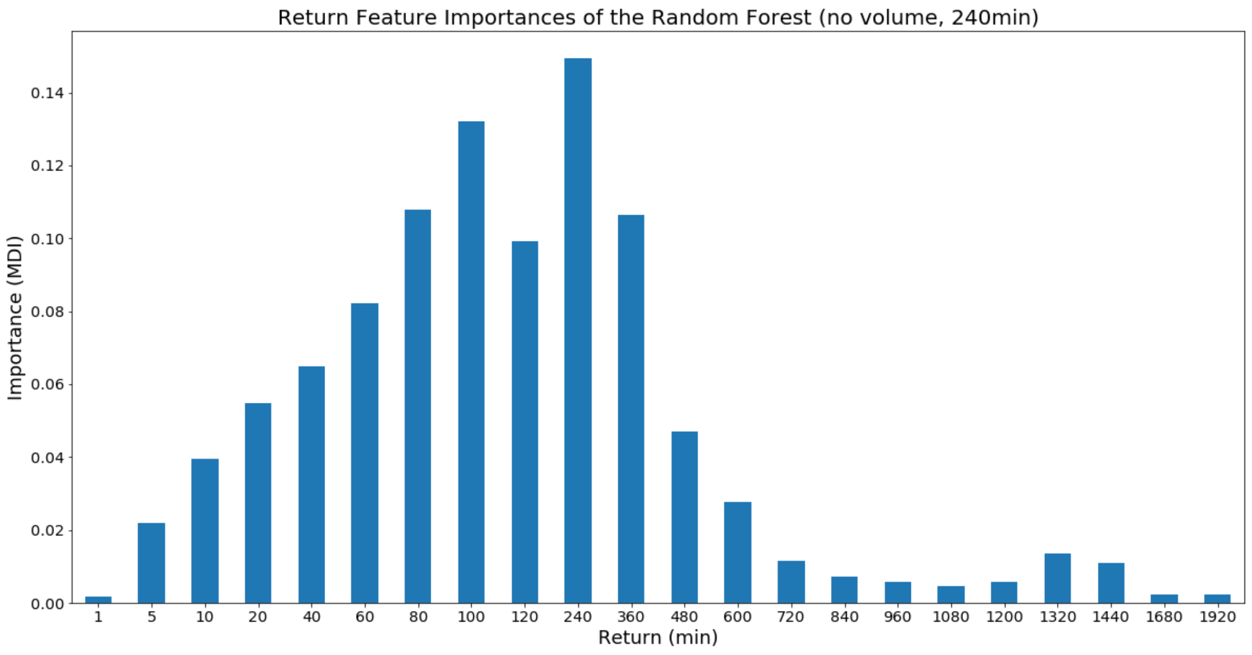


Figure 12: This figure illustrates the feature importances of the Random Forest according to MDI [28] for different return deltas in minutes as described in chapter 4.1. The Random Forest was trained on features containing only returns and a forecast duration of 240min.

Extending the duration to 240min leads to similar feature importance pattern as can be seen in figure 12. However, including volume in the feature selection leads to a radically different shape of feature importances as can be seen below in figure 13.

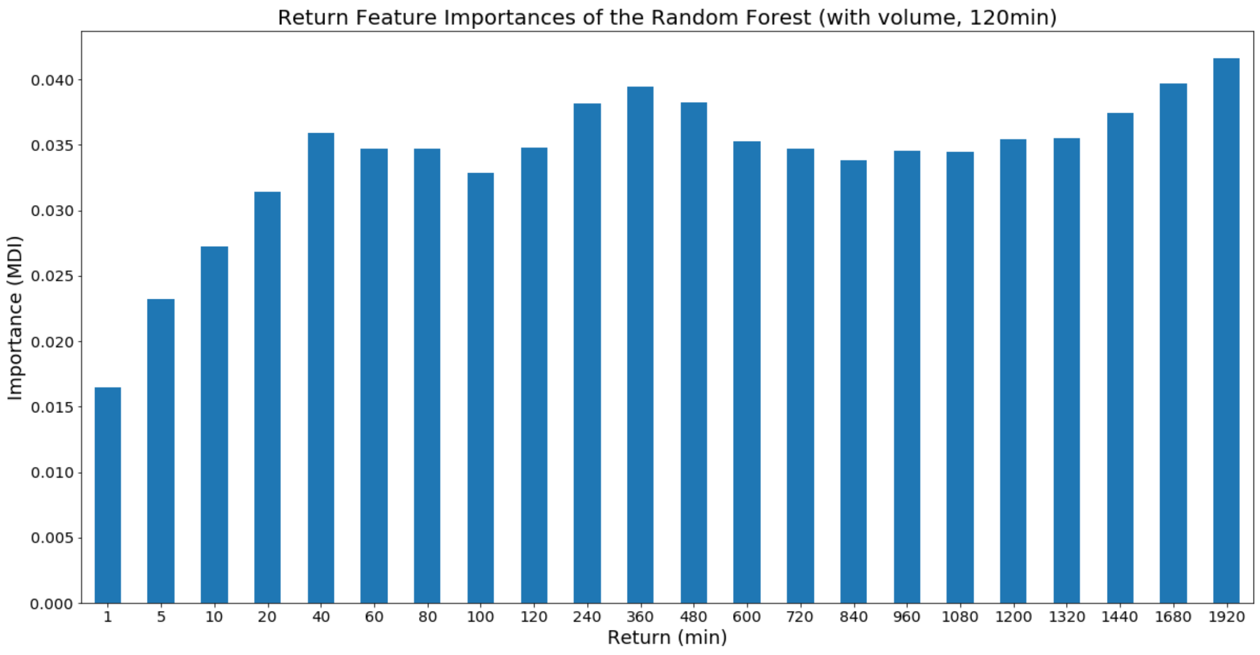


Figure 13: This figure illustrates the feature importances of the Random Forest according to MDI [28] for different return deltas in minutes as described in chapter 4.1. The Random Forest was trained on features containing both returns and volumes and a forecast duration of 120min.

In addition, also the MDI values are considerably lower than those in the graph without using volumes for the model fit. This indicates that the inclusion of volumes changes the relationship of returns and time for the Random Forest. This is consistent across different forecast durations (see appendix figure 20).

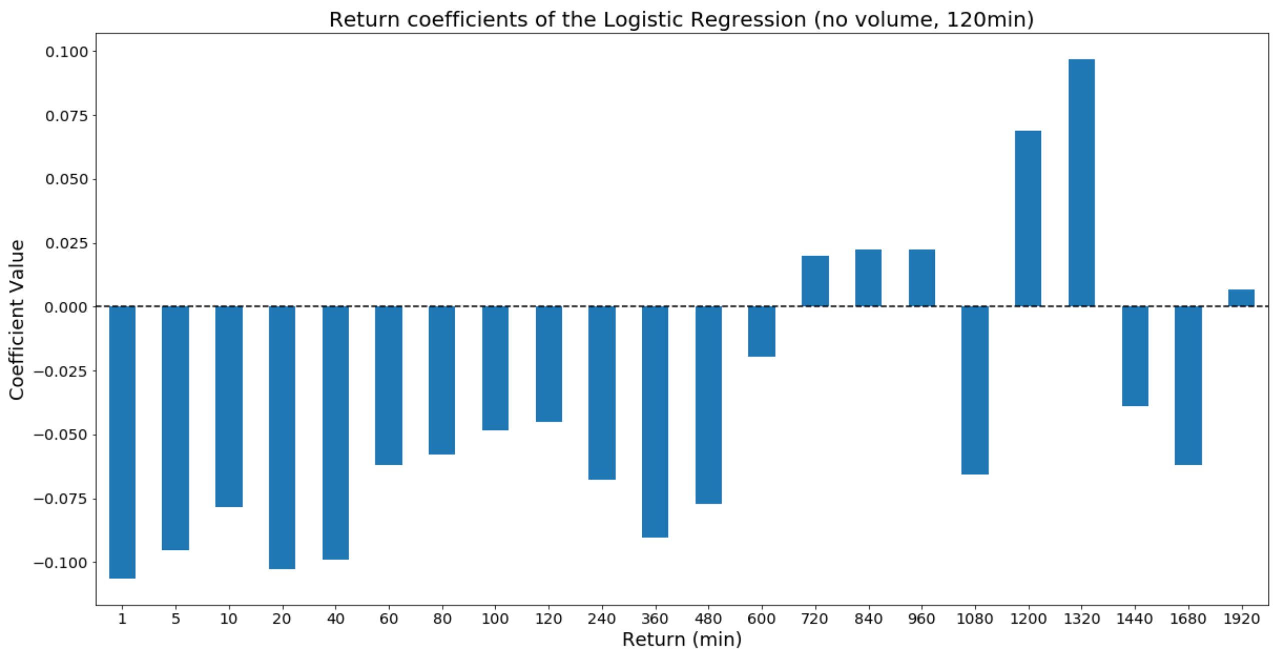


Figure 14: This figure illustrates the coefficients for the Logistic Regression for different return deltas in minutes as described in chapter 4.1. The Logistic Regression was trained on features containing only returns and a forecast duration of 120min.

Figure 14 illustrates the coefficients of the Logistic Regression for different return deltas in minutes. Comparing the absolute coefficient values of with the MDI values from the Random Forest [28], the relationship of past return on forecasted future returns seems to be inverted. The highest coefficient values are realized at the beginning and near the end which is in stark contrast to the Random Forest. Also this finding is consistent across specifications (see appendix figure 21), which was not the case for the Random Forest. Due to the high interpretability of the Logistic Regression, one can also conclude that high returns in the first 600 min lead to low probability of an up-classification. Thus, one can cautiously conclude that the Logistic Regression puts an emphasis on price stability, because it implicitly supposes a revertion of trend (down-movement after a series of positive returns and up-movement after a series of negative returns).

6 Conclusion

Considering the results from above, I conclude that it is not feasible to consistently identify arbitrage opportunities with this setup solely using OHLC price data. Even when fine tuning the probability threshold for each model, most of them fail to generate any positive return over the trading period. Also, even the best performance is outmatched by a simple buy-and-hold equal-weight portfolio. Without fine tuning the probability threshold for each model, the negative returns reach values below -60% (see appendix 19). Thus, I conclude that the results are supportive of the first efficient market hypothesis [10].

One could further test this hypothesis on crypto markets by introducing a target variable containing a third state which indicates that returns do not go beyond transaction cost. This would help avoid unnecessary trading decisions. Further, one could focuse more on the market microstructure, especially in the context of high frequency trading. For example, one could use or even develop models which can interpret the occurence of missing bins accordingly. In this application, due to the high trading frequency, the trading strategy is relatively often confronted with liquidity issues leading to missing bins. In addition, one could use agent based models with profits as reward such as Reinforcement Learning. In theory, they would learn how to interact properly with the market, and even take the influence of their own actions into account. This could be a quite valuable property, since it is possible to influence markets with large enough orders, due to the unregulated nature of crypto markets and low market actvity and volume.

References

[1] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* (2011).

[2] Raman Arora et al. “Understanding Deep Neural Networks with rectified linear units”. In: *ICLR 2018* (2018).

[3] Julius Berner et al. “Towards a regularity theory for ReLU networks – chain rule and global error estimates”. In: *13th International conference on Sampling Theory and Applications* (2019).

[4] L. Breiman et al. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

[5] *CoinMarketCap*. 2013. URL: https://coinmarketcap.com/de/exchanges/bitfinex/ (Accessed July 3, 2020).

[6] Thomas G Dietterich. “An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization”. In: *Machine Learning 40* (2000).

[7] Matthew Dixon, Diego Klabjan, and Jin Hoon Bang. “Implementing Deep Neural Networks for Financial Market Prediction on the Intel Xeon Phi”. In: *Proceedings of the 8th Workshop on High Performance Computational Finance* (2015).

[8] Anne Haubo Dyhrberg. “Bitcoin, gold and the dollar – A GARCH volatility analysis”. In: *Finance Research Letters* (2015).

[9] Mark David Enke and Suraphan Thawornwong. “The use of data mining and neural networks for forecasting stock market returns”. In: *Expert Systems with Applications* (2005).

[10] Eugene F. Fama. “Efficient Capital Markets: A Review of Theory and Empirical Work”. In: *The Journal of Finance* (1970).

[11] Thomas Günter Fischer, Christopher Krauss, and Alexander Deinert. “Statistical Arbitrage in Cryptocurrency Markets”. In: *Journal of Risk and Financial Management* (2019).

[12] Thomas Fischer and Christopher Krauss. “Deep learning with long short-term memory networks for financial market predictions”. In: *FAU Discussion Papers in Economics* (2017).

[13] Yoav Freund and Robert E. Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Science* (1997).

[14] Yoav Freund and Robert E. Schapire. "A Short Introduction to Boosting". In: *Machine Learning: Proceedings of the Thirteenth International Conference* (1996).

[15] Yoav Freund and Robert E. Schapire. "A Short Introduction to Boosting". In: *Journal of Japanese Society for Artificial Intelligence* (1999).

[16] Jerome H. Friedman. "Stochastic Gradient Boosting". In: *Computational Statistics & Data Analysis 38* (2000).

[17] Evan Gatev, William N. Goetzmann, and K. Geert Rouwenhorst. "Pairs Trading: Performance of a Relative Value Arbitrage Rule". In: *The Review of Financial Studies* (2006).

[18] Stuart Geman, Elie Bienenstock, and René Doursat. "Neural networks and the bias/variance dilemma". In: *Neural computation* (1992).

[19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Berlin Heidelberg, 2009.

[20] Robby Houben and Alexander Snyers. "Crypto-assets: Key developments, regulatory concerns and responses". In: *European Parliament's Committee on Economic and Monetary Affairs* (2020).

[21] Nicolas Huck. "Pairs selection and outranking: An application to the S&P 100 index". In: *European Journal of Operational Research* (2009).

[22] iFinex Inc. *Bitfinex*. 2012. URL: https://www.bitfinex.com/ (Accessed July 3, 2020).

[23] Gareth James et al. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013.

[24] Hamid Karimi, Tyler Derr, and Jiliang Tang. "Characterizing the Decision Boundary of Deep Neural Networks". In: *arXiv* (2019).

[25] Nittaya Kerdprasop and Kittisak Kerdprasop. "Discrete decision tree induction to avoid overfitting on categorical data". In: *13th WSEAS International Conference on Systems Theroy and Scientific Computation* (2013).

[26] Christopher Krauss, Xuan Anh Do, and Nicolas Huck. “Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500”. In: *Econstor* (2016).

[27] Mark T. Leung, Hazem Daouk, and An-Sing Chen. “Forecasting stock indices: a comparison of classification and level estimation models”. In: *International Journal of Forecasting* (2000).

[28] Gilles Louppe et al. “Understanding variable importance in forests of randomized trees”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems* (2013).

[29] Burton G. Malkiel. “The Efficient Market Hypothesis and its Critics”. In: *Journal of Economic Perspectives* (2003).

[30] Sean McNally, Jason Roche, and Simon Caton. “Predicting the Price of Bitcoin Using Machine Learning”. In: *26th Euromicro International Conference on Parallel, Distributed and Network-based Processing* (2018).

[31] Benjamin Moritz and Tom Zimmermann. “Deep conditional portfolio sorts: The relation between past and future stock returns”. In: *Working Paper* (2014).

[32] Alexandru Niculescu-Mizil and Rich Caruana. “Predicting Good Probabilities With Supervised Learning”. In: *Proceedings of the 22nd international conference on Machine learning* (2005).

[33] *NumPy Documentation*. 2020. URL: https://numpy.org/doc/ (Accessed July 3, 2020).

[34] *NumPy Documentation*. 2020. URL: https://matplotlib.org/ (Accessed July 3, 2020).

[35] *pandas documentation*. 2020. URL: https://pandas.pydata.org/docs/ (Accessed July 3, 2020).

[36] Hyeoun-Ae Park. “An Introduction to Logistic Regression: From Basic Concepts to Interpretation with Particular Attention to Nursing Domain”. In: *J Korean Acad Nurs* (2013).

[37] *Python 3.7.8 documentation*. 2020. URL: https://docs.python.org/3.7/ (Accessed July 3, 2020).

[38] Raul Rojas. *Neural Networks: A Systematic Introduction*. Springer Berlin Heidelberg, 1996.

[39] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *ArXiv* (2016).

[40] Leszek Rutkowski et al. “The CART Decision Tree for Mining Data Streams”. In: *Information Sciences Volume 266* (2013).

[41] *scikit-learn*. 2020. URL: https://scikit-learn.org/stable/ (Accessed July 3, 2020).

[42] Devavrat Shah and Kang Zhang. “Bayesian regression and Bitcoin”. In: *Annual Allerton Conference on Communication, Control, and Computing* (2014).

[43] Lawrence Takeuchi and Yu-Ying (Albert) Lee. “Applying Deep Learning to Enhance Momentum Trading Strategies in Stocks”. In: *Working Paper, Stanford University* (2013).

[44] Bianca Zadrozny and Charles Elkan. “Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers”. In: *Proceedings of the 18th International Conference on Machine Learning* (2001).

[45] Jiawei Zhang. “Basic Neural Units of the Brain: Neurons, Synapses and Action Potential”. In: *arXiv* (2019).

[46] Peter G. Zhang, Eddy Patuwo, and Michael Y. Hu. “Forecasting With Artificial Neural Networks: The State of the Art”. In: *International Journal of Forecasting* (1998).

[47] Ji Zhu et al. “Multi-class AdaBoost”. In: *Statistics and Its Interface* (2009).

7 Appendix

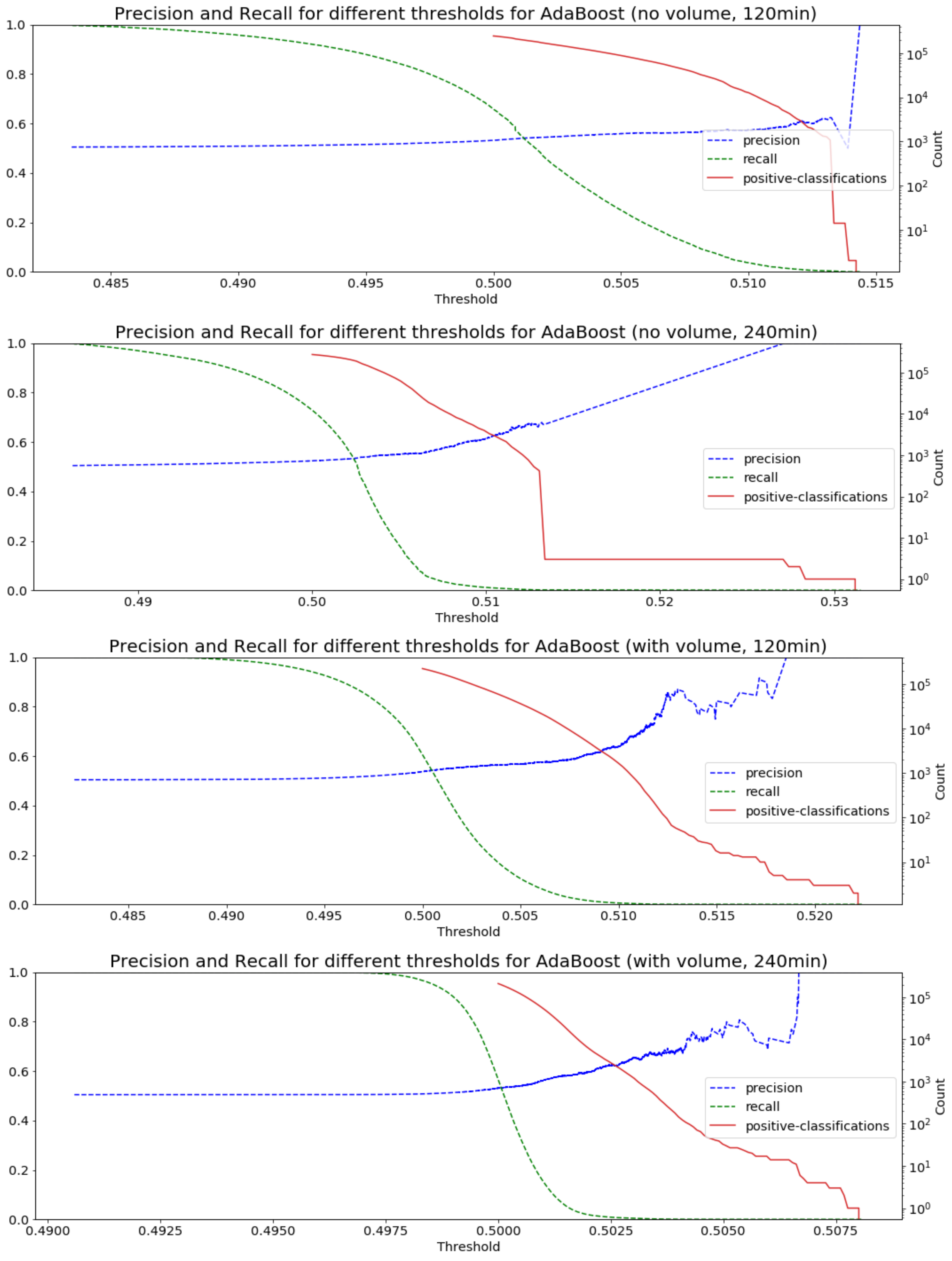


Figure 15: This figure illustrates the precision, recall and up-classifications of the AdaBoost for different thresholds and specifications.

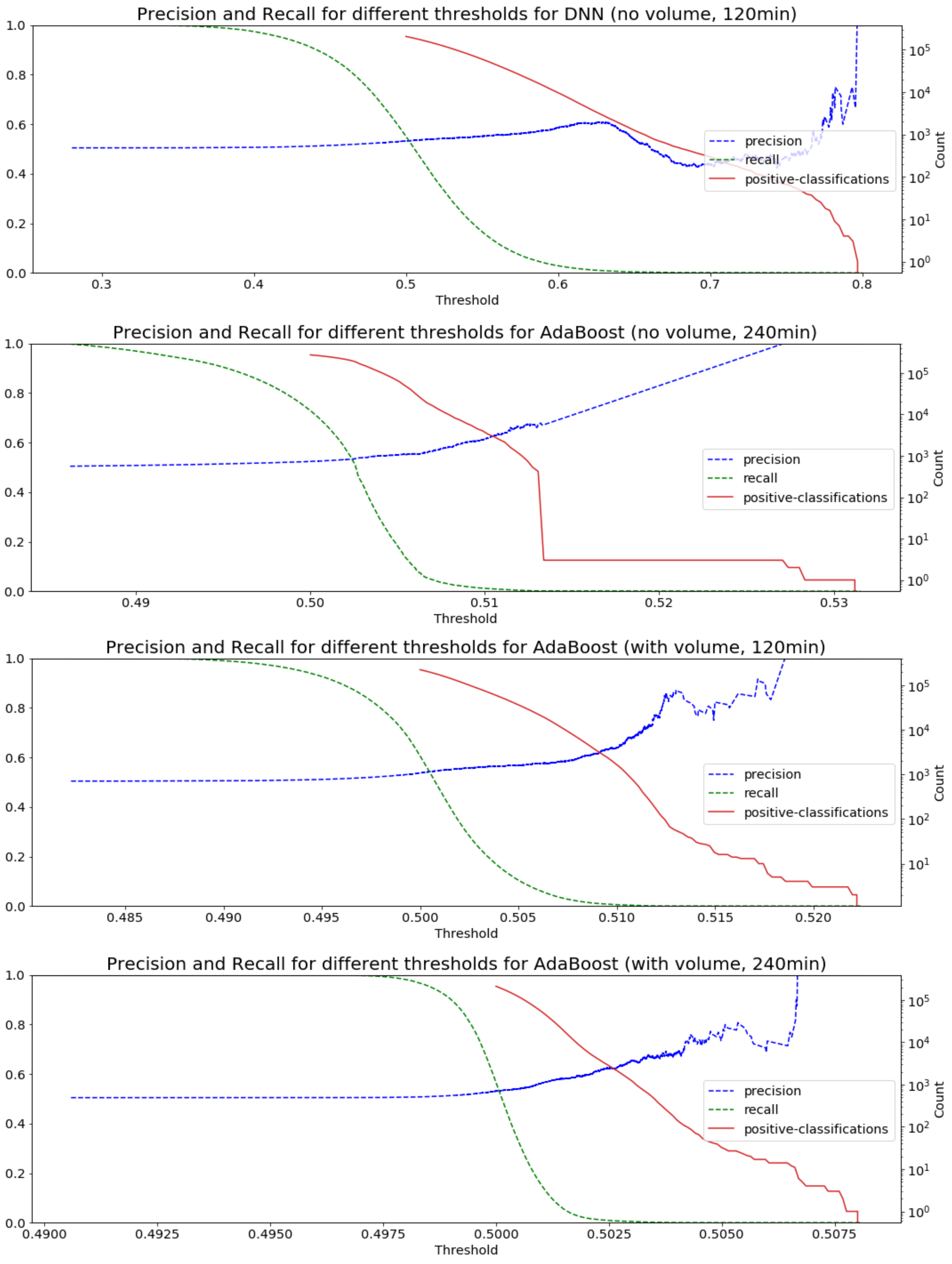


Figure 16: This figure illustrates the precision, recall and up-classifications of the Deep Neural Network for different thresholds and specifications.

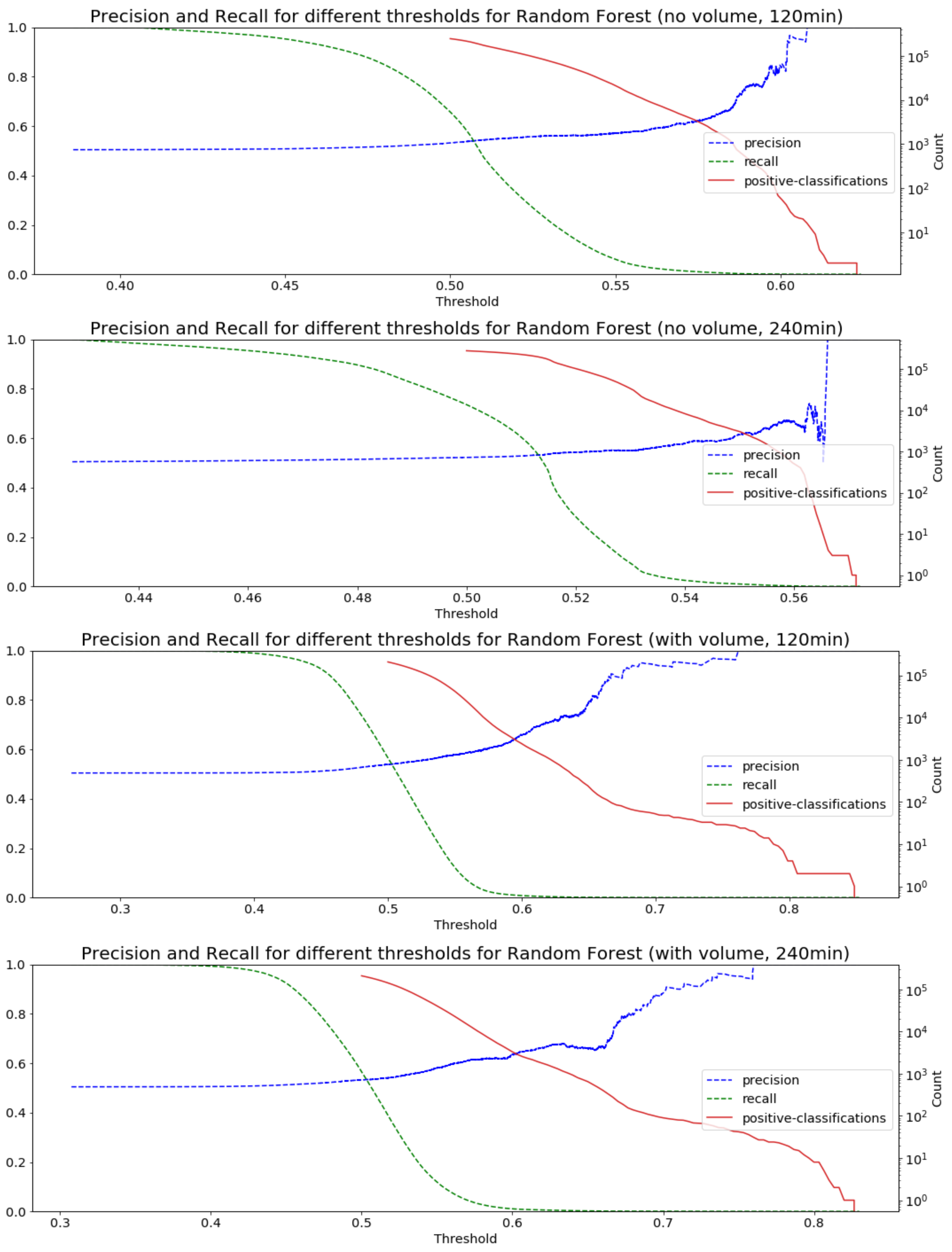


Figure 17: This figure illustrates the precision, recall and up-classifications of the Random Forest for different thresholds and specifications.

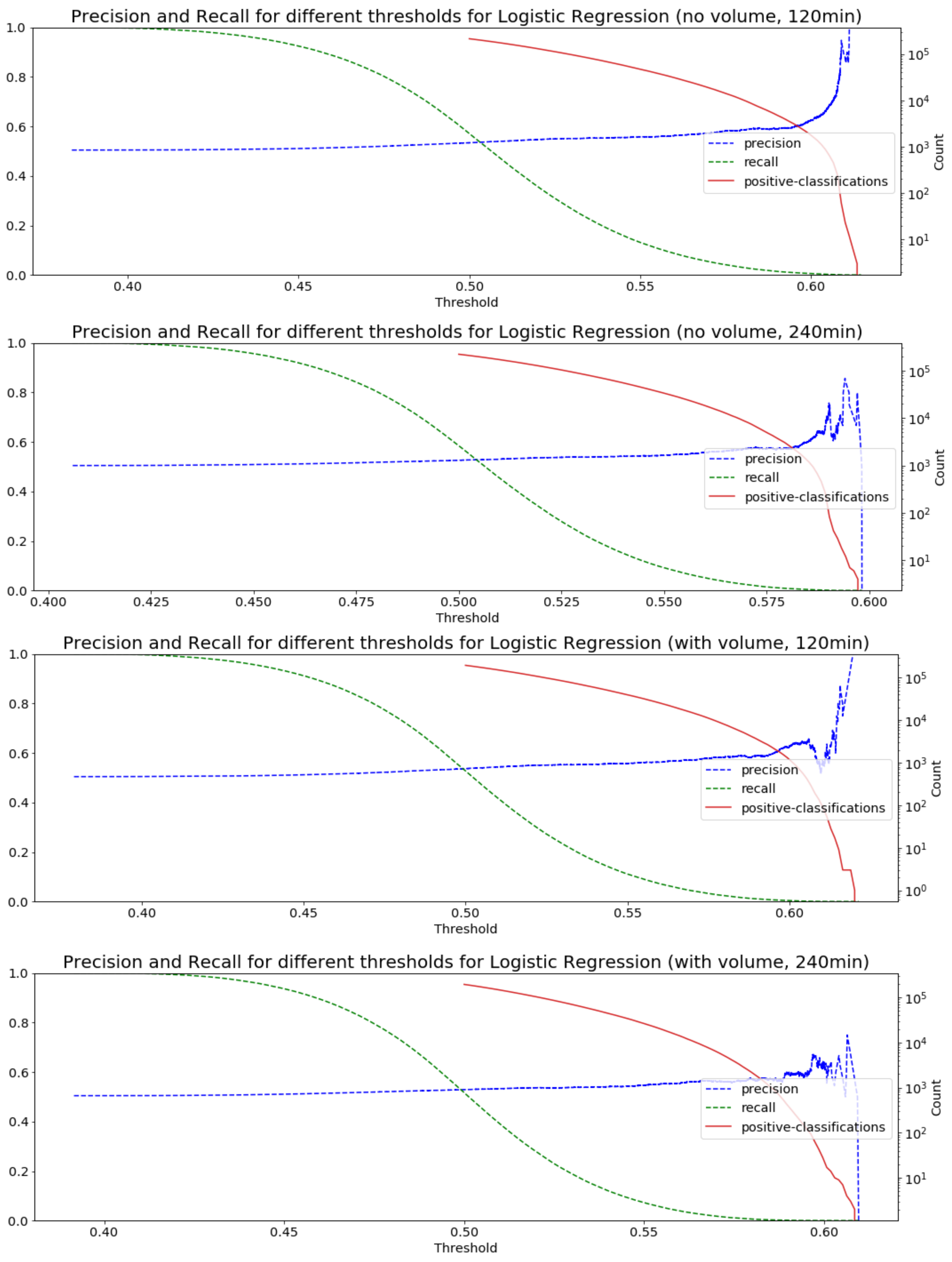


Figure 18: This figure illustrates the precision, recall and up-classifications of the Logistic Regression for different thresholds and specifications.

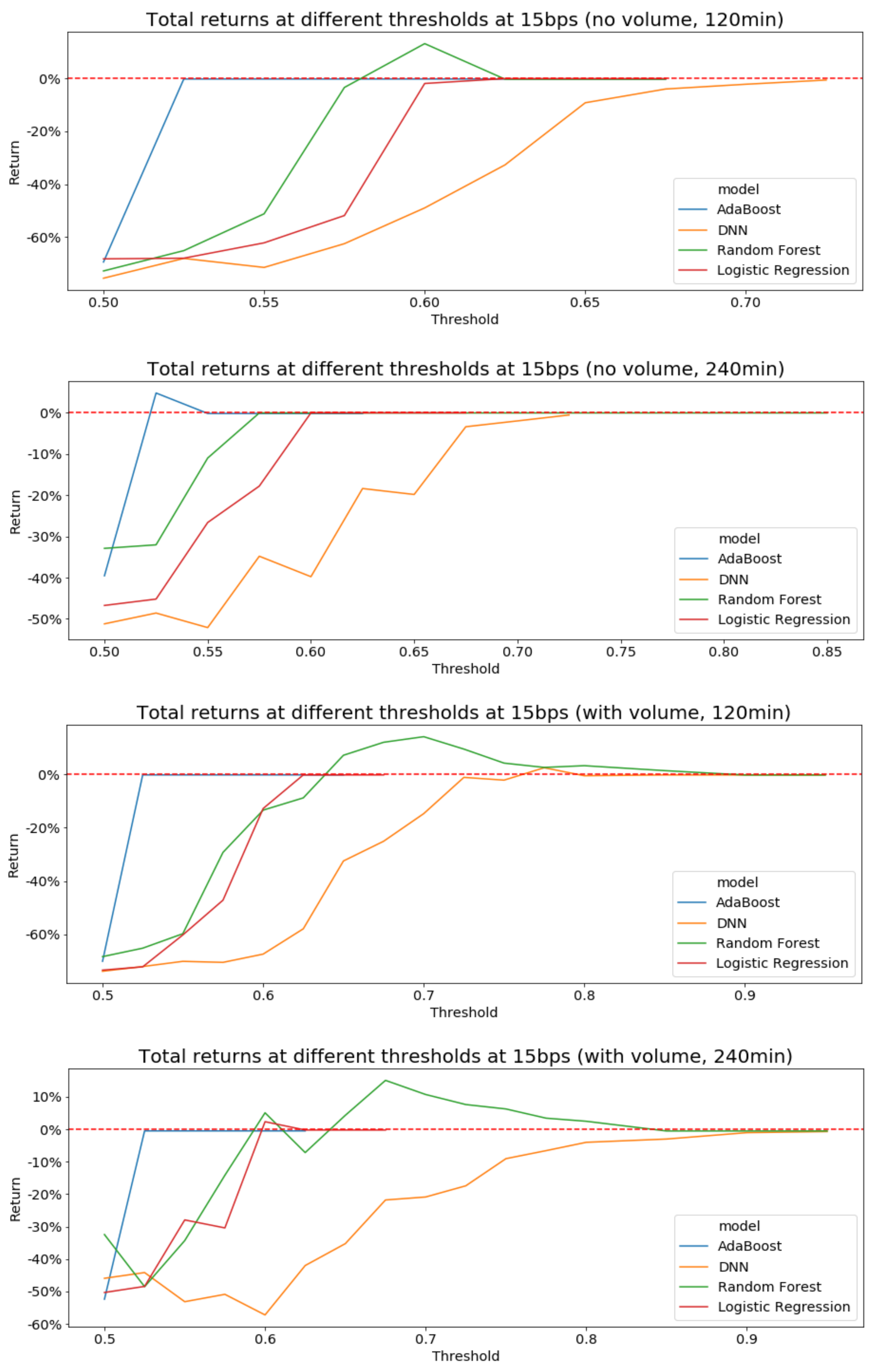


Figure 19: This figure illustrates total returns at a transaction cost of 15bps over the trading period for different thresholds and specifications.

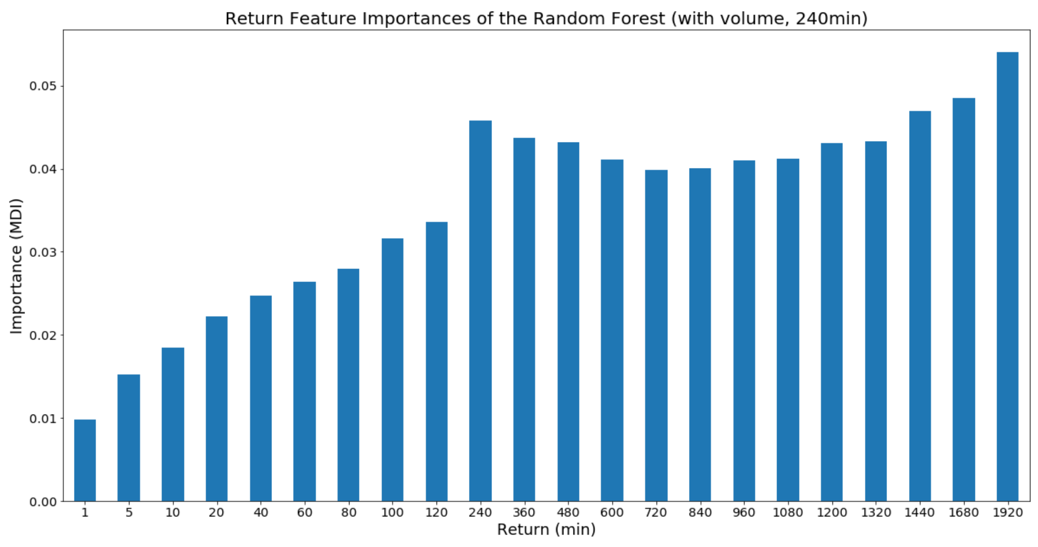


Figure 20: This figure illustrates the feature importances of the Random Forest according to MDI [28] for different return deltas in minutes as described in chapter 4.1. The Random Forest was trained on features containing both returns and volumes and a forecast duration of 240min.

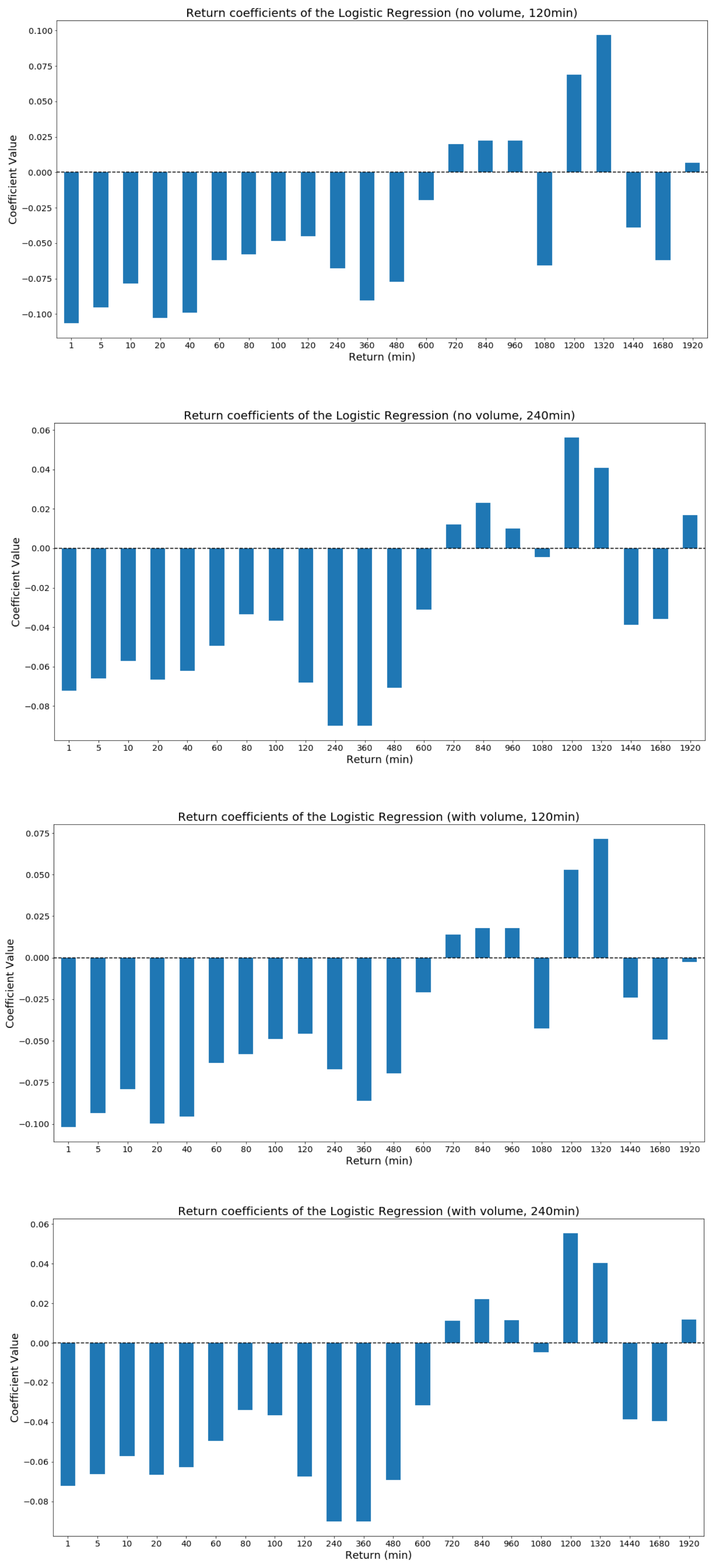
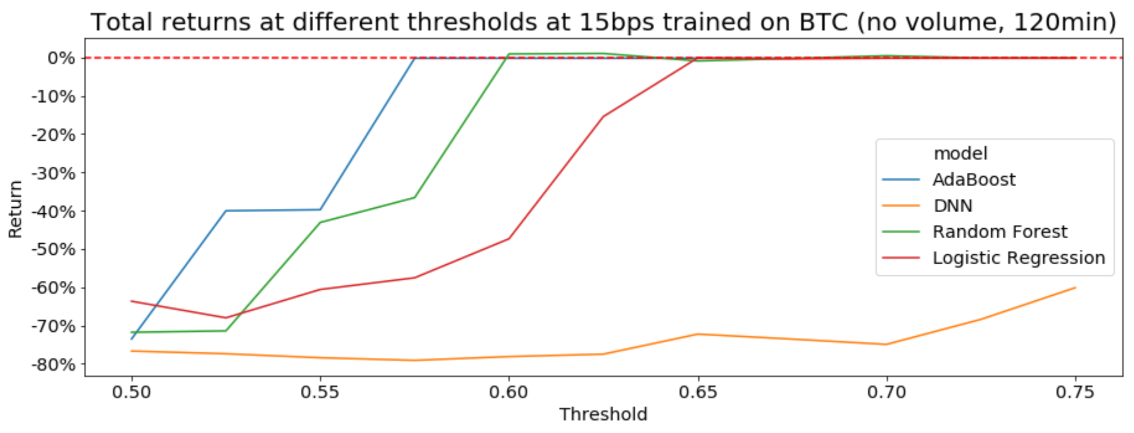
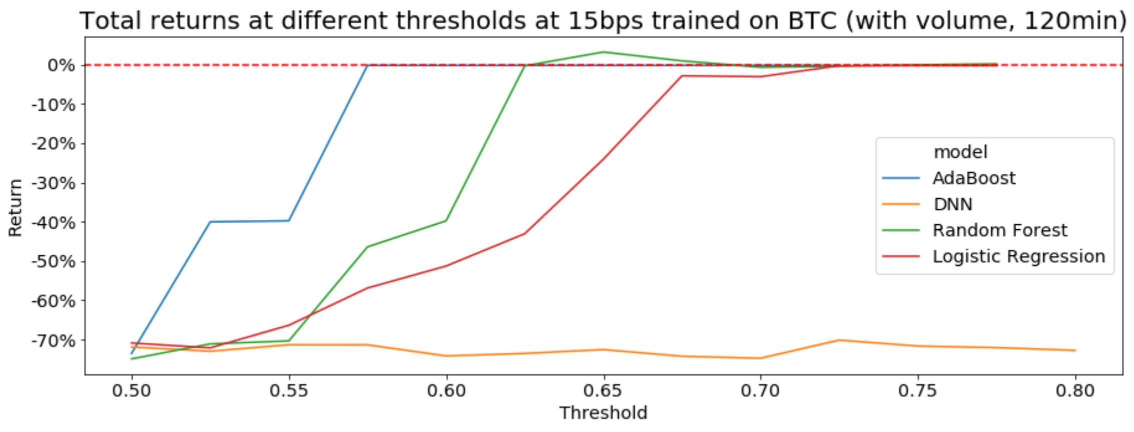
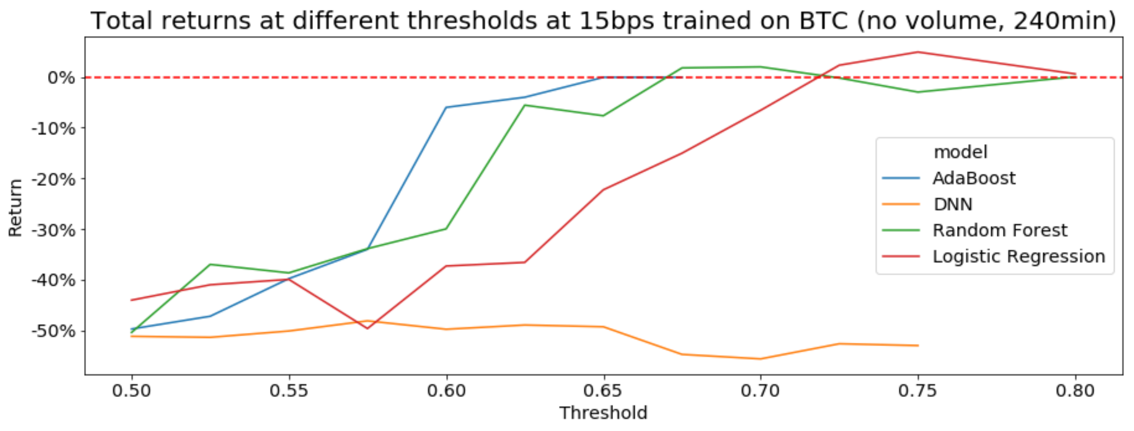


Figure 21: This figure illustrates the coefficients of returns estimated by the Logistic Regression for different time periods in minutes for different feature and target specification.







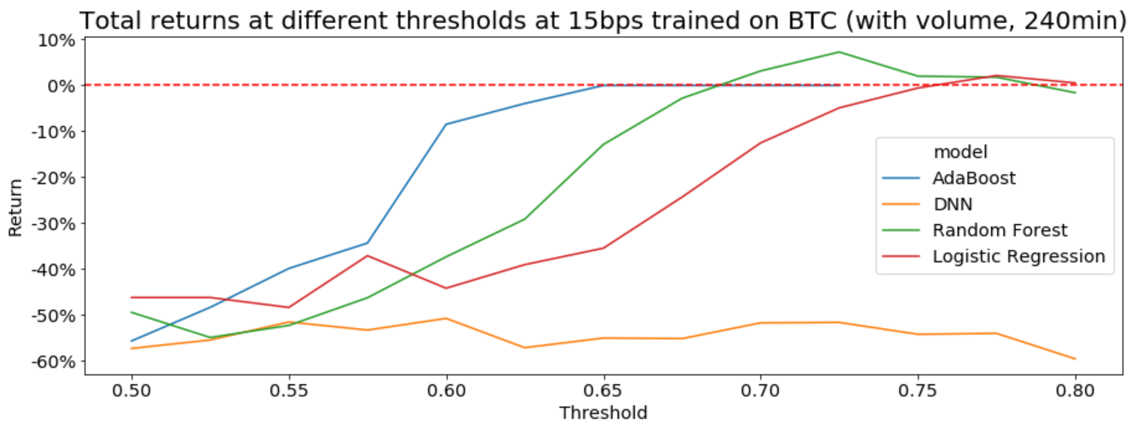


Figure 22: This figure illustrates total returns at a transaction cost of 15 bps over the trading period for different thresholds across different feature and target specifications. The four models were trained on features only containing returns and a forecast duration of 120 min from BTC data.

”I hereby confirm that the work presented has been performed and interpreted solely by myself except for where I explicitly identified the contrary. I assure that this work has not been presented in any other form for the fulfillment of any other degree or qualification. Ideas taken from other works in letter and in spirit are identified in every single case.”

Bonn, August 27, 2020