University of Bonn

Master of Science in Economics

Identifying Arbitrage in Cryptomarkets with Algorithmic Trading:
A Machine Learning Approach

Submitted by

Raphael Redmer

Supervisor: Prof. Dr. Joachim Freyberger

July 14, 2020

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

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 [14]. Moreover, the generally accepted weak efficient market hypothesis provides a theoretical framework which encompasses this phenomenon [12]. The weak form suggests that current asset prices reflect all the information of past prices and that no form of technical analysis can be effectively utilized to aid investors in making trading decisions, thus eliminating arbitrage solely based on price data. So in theory, also machine learning algorithms which were trained on price data should not be able to do so. In order to test this hypothesis, we train multiple machine learning models on price data with different specifications to predict price movements. We chose a classification instead of a regression problem, as the literature suggests that the former performs better than the latter in predicting financial market data ([28], [11]). Then, we use the estimated probabilities from the models in a backtest for a trading algorithm to simulate actual trading. 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 ([10], [22]). 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 we expect that arbitrage opportunities only rarely occur. This expectation is further supported by low entry barriers and transaction costs [24].

TODO: Foreshadow Results and specify contribution to literature

2 Literature Review

One of the first works addressing this question is [42]. More specifically, the authors tried to predict price changes of Bitcoin during a six month period in 2014 with a Bayesian regression model. The results are astonishing, with a return of 89 percent and a Sharpe ratio of 4.10 during a period of merely 50 trading days. However, no transaction costs are taken into account, perfect liquidity is assumed, and only one cryptocurrency is considered. [43] develop an enhanced momentum strategy on the U.S. CRSP stock universe from 1965 until 2009. Specifically, deep neural networks are employed as classifiers to calculate the probability for each stock to outperform the cross-sectional median return of all stocks in the holding month $t+1$. [9] run a similar strategy in a high-frequency setting with five-minute binned return data. They reach substantial classification accuracy of 73 percent, albeit without considering microstructural effects - which is essential when dealing with high-frequency data. [32] deploy random forests on U.S. CRSP data from 1968 to 2012 to develop a trading strategy relying on "deep conditional portfolio sorts". Specifically, they use decile ranks based on all past one-month returns in the 24 months prior to portfolio formation at time t as predictor variables. A random forest is trained to predict returns for each stock s in the 12 months after portfolio formation. The top decile is bought and the bottom decile sold short, resulting in average risk-adjusted excess returns of 2 percent per month in a four-factor model similar to [6]. Including 86 additional features stemming from firm characteristics boosts this figure to a stunning 2.28 percent per month. Highest explanatory power can be attributed to most recent returns, irrespective of the inclusion of additional firm characteristics. In spite of high turnover, excess

returns do not disappear after accounting for transaction costs. [13] train a random forest on lagged returns of 40 cryptocurrency coins, with the objective to predict whether a coin outperforms the cross-sectional median of all 40 coins over the subsequent 120 min. They buy the coins with the top-3 predictions and short-sell the coins with the flop-3 predictions, only to reverse the positions after 120 min. During the out-of-sample period of our 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. [27] implement and analyse the effectiveness of deep neural networks, gradientboosted-trees, random forests, and a combination of these methods in the context of statistical arbitrage. Each model is trained on lagged returns of all stocks in the S&P 500, after elimination of survivor bias. From 1992 to 2015, daily one-day-ahead trading signals are generated based on the probability forecast of a stock to outperform the general market. The highest probabilities are converted into long and the lowest probabilities into short positions, thus censoring the less certain middle part of the ranking.

3 Data and Software

3.1 Data

In this setup, we use minute-binned OHLC data of crypto/USD-pairs obtained from the cryptocurrency exchange Bitfinex via its API ranging from 01.01.2019 to 31.12.2019. For each minute-bin, we collect *Open*, *High*, *Low*, *Close*, *Volume* and *Timestamp* data. *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 roughly 111.366.484 USD of and 155 different Dollar-tradable coins [24], 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).

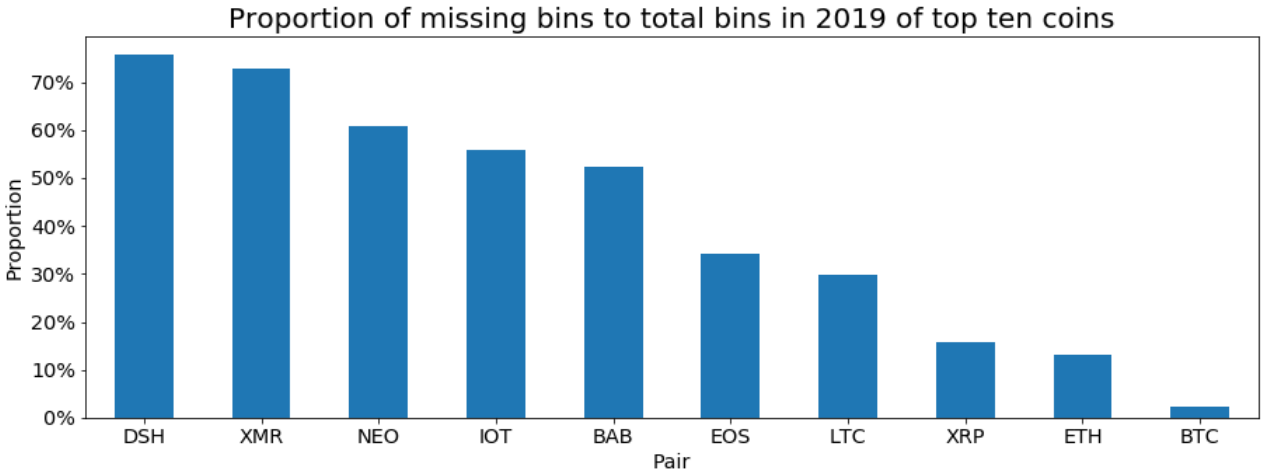

Figure 1: This figure illustrates the proportion of missing minute-bins to total number of minute-bins in 2019 for the top ten coins.

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. We resolved this issue by propagating price values from the last active minute-bin and setting the volume to zero. Further, we restricted the number of crypto/USD-pairs to the top ten pairs by market capitalization [7]. In addition, we decided to only take data from 01.01.2019 to 31.12.2019, since for most coins 2019 was the most active year in terms of trading frequency. 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, we used Pandas and numpy ([35], [33]). Data Visualization was done via Maplotllib [34]. For the training of the models Logistic Regression, Random Forest and Support Vector Machine, we used the respective Scikit-learn implementation [41]. The Artificial Neural Network was trained using the Keras framework with Tensorflow backend to enable GPU calculation.

4 Methodology

Similarly to [27] and [13], the methodology of this paper consists of the following steps:

1. The entire data set is split into a training, a validation and a trading set.
2. The respective features (explanatory variables) and targets (dependent variables) are created
3. Each model is trained on the training set
4. Conduct out-of-sample predictions 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 our 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 the valdation set. We 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.

4.2 Feature and Target Generation

Broadly following [43], we generate the feature space as follows:

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

Features: From the data set we obtain the following 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_t^c be the traded volume for coin c in minute-bin t scaled by Quantile-Transformer fitted separately for each coin only on bins with $V_t^c > 0$.

Target: Let $Y^c_{t+1,t}$ be a binary response variable for each coin c and $d = 120$ the size of the future time interval. It assumes value 1 (class *up*) if its future 120 min return $R^c_{t+d,t+1}$ is greater than its cross-sectional median across all pairs $(R^c_{t+d,t+1})^n_{c=1}$, else -1 (class *down*). Instead of just using the simple return $R^c_{t+d,t+1}$ as in [13], we included an additional condition, which demands that $V^c_{t+d} > 0$ for realizing the feature return. If not skip bins until you reach a bin $t^* = t + d + \delta$ 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 we only allow trades to be executed in a bin, if any volume was traded for the respective coin. We 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 ([31], [12]). In addition, the volume got scaled for each coin in order to make the measure more comparable across coins, since we are training a single universal model for each of the selected coins. Further, the Quantile-Transformation handles outliers (¡cite¿) and restricts the feature to an intervall 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 we denote diffenrentiate 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 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,

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

There

4.3 Model Training

As explained in chapter 4.1, we construct a training set ranging 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, we restrict the training and validation set by excluding bins for which no volume was traded in the following bin or lagged values are not available.

We cross-validate the respective parameter space by first training the model on the test set an evaluating on the chronologically following validation set for each parameter combination. After obtaining an accuracy evaluation of each combination, we fit the model with the best validation performance on the combined training and valdation set.

Further, we also give a brief description of the models and how we used them for our trading experiments. Before doing so, we introduce the mathematical notation. Let $D = \{(x_i, y_i)\}_{i=1}^n$ denote training data set containing n observations on which the respective model gets fitted whereas $x_i \in \mathbb{R}^K$ the denotes feature vector with K features and y_i the target class variable for the i-th observations respectively. In addition, we denote a Decision Tree [5] as a function $T^q_D(x)$ with D denoting the set which is was trained on, q the selection of features it was trained on and x feature vector as an the function argument. Decision Trees $T^q_D(x)$ are used in Random Forest and AdaBoost.

4.3.1 Logistic Regression

Logistic Regression sometimes called the logistic model or logit 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 our application, since the dependent variable is dichotomous (either *down*- or *up*-movement, i.e. 0 or 1).

The building block for the Logistic Regression is the multiple regression model ([25], [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 $p(X)$. Any time a straight line is fit to a binary response that is coded as 0 or 1, in principle we can always predict $p(X) < 0$ for some values of X and $p(X) > 1$ for others (unless the range of X is limited) (¡intro statistics¿). Therefore, to avoid this problem in the Logistic Regression, we wrap the *logistic function* 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, we can use the *maximum likelihood* method. The basic intuition behind using maximum likelihood to fit a logistic regression model is as follows: we seek 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 intuition can be formalized using a mathematical equation called a *likelihood function*:

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

Therefore, we have the following optimization problem for the Logistic Regression:

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

We 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 ([14], [13]). For this paper, we rely on the Scikit-learn implementation of [1] for the logistic regression and follow the parameters outlined in [14], i.e., the optimal L2-regularization is determined among 100 values on a logarithmic scale from 0.0001 to 10,000 via cross-validation explained above on the respective training set, and L-BFGS is deployed to find an optimum. Further, we restrict the maximum number of iterations to 100.

4.3.2 Random Forest

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

The Decision Tree is a non-parametric supervised 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 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.

An example of such a tree is depicted below in figure 2.

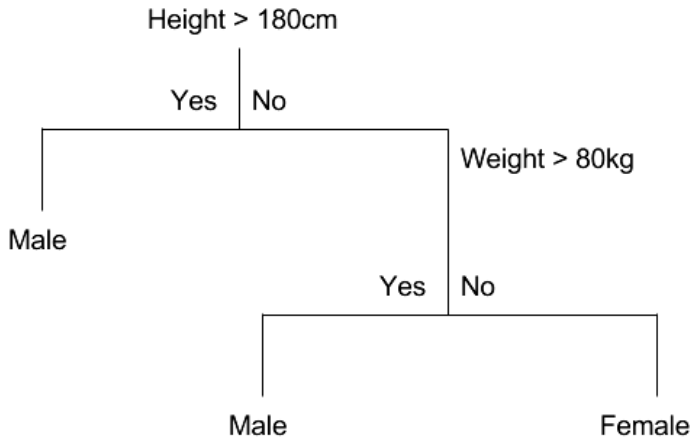

Figure 2: Given a data set with two features height and weight, and gender as the target variable, this example tree stratisfies the two-dimensional feature space into three distinct subset each represented by the terminal nodes at the bottom. The stratification occurs at the two deciding nodes depending either on whether its height is above 180 cm and its weight is above 80kg.

In the following, we describe the CART algorithm for tree building as specified in [5]. 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 [5]

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 ([26], [20]), 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 addition 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 a suitable candidate for predicting price movements lies in its ability to model non-linear decision boundaries and to grow arbitrarily complex models. In addition, Random Forests in this configuration are the best single technique in [27] and the method of choice in [32] - a large-scale machine learning application on monthly stock market data. As such, random forests serve as a powerful benchmark for any innovative machine learning model.

In our application of the Random Forest, we 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 [27], [13] and [1].

4.3.3 AdaBoost

AdaBoost (Adaptive Boosting) classifier is a meta-estimator that begins by fitting a classifier, a Decision Tree in our application, on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases [15]. 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* [45]. These are individual classification models wich have by itself a small predictive power. In our application, we use small Decision Trees with a depth of up to three. Starting with the unweighted training sample, the AdaBoost builds a classifier, for example a classification tree, that produces class labels. If a training data point is misclassified, the weight of that training data point is increased (boosted). A second classifier is built using the new weights, which are no longer equal. Again, misclassified training data have their weights

boosted and the procedure is repeated. The AdaBoost algorithm is illustrated below:

Algorithm 3: AdaBoost [45]

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` α `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 $C(x) = \arg\max_k \sum_{m=1}^M \alpha^{(m)} \cdot \mathbb{1}(T_{\widetilde{D}}^{(m)}(x_i) = k)$

Practically, AdaBoost has many advantages. It is fast, simple and easy to program and it 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. Finally, it comes with a set of theoretical guarantees given sufficient data and a weak learner that can reliably provide only moderately accurate weak hypotheses. This is a shift in mind set for the learning-system designer: instead of trying to design a learning algorithm that is accurate over the entire space, we can instead focus on finding weak learning algorithms that only need to be better than random [17]. A nice property of AdaBoost is its ability to identify outliers, i.e., examples that are either mislabeled in the training data, or which are inherently ambiguous and hard to categorize. Because AdaBoost focuses its weight on the hardest examples, the examples with the highest weight often turn out to be outliers ([16], [17]). 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, consistent with theory, boosting can fail to perform well given insufficient data, overly complex weak hypotheses or weak hypotheses which are too weak. Boosting seems to be especially susceptible to noise ([17], [8]). Therefore, this experiment shows how this tradeoff will be resolved. In our application of the AdaBoost, we 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 [27], [21] and [18].

4.3.4 Deep Neural Network

In this chapter, we briefly describe the Deep Neural Network (DNN) according to [9] and [27]. A deep neural network consists of an input layer, one or more hidden layers, and an output layer, forming the topology of the net as can be seen in figure 3.

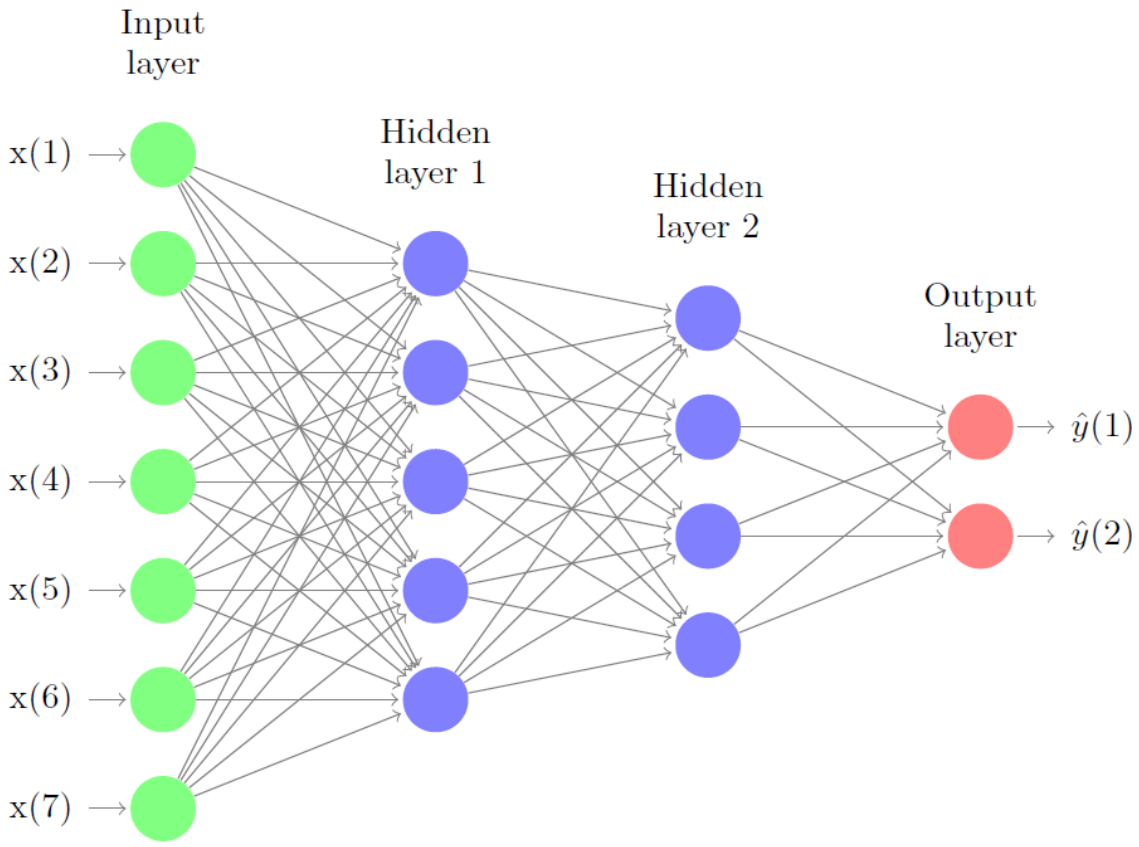

Figure 3: An illustrative example of a feed-forward neural network with two hidden layers, seven fea- tures and two output states. Deep learning networks typically have many more layers, use a large number of features and several output states or classes. The goal of learning is to find the weight on every edge that minimizes a loss function.

The input layer matches the feature space, so that 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 are composed of neurons, the basic units of such a model. In the classical feedforward architecture, each neuron in the previous layer l is fully connected with all neurons in the subsequent layer $l+1$ via directed edges, each representing a certain weight. Also, each neuron in a non-output layer of the net has a bias unit, serving as its activation threshold. As such, each neuron j of layer l receives a weighted combination α 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 x_i^{l-1} 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 via some activation function f, so that the output signal $f(\alpha)$ is relayed 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. We use the ReLU (rectified linear activation unit) ([2], [3]) $f : \mathbb{R} \to \mathbb{R}$,

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

Let W be the collection $W = \bigcup_{l=1}^{L-1} W_l$, with $W_l = (w_{i,j}^l) \in \mathbb{R}^{n_{l-1} \times n_l}$ denoting the weight matrix that connects layers l and $l+1$ for a network of L layers. Analogously, let B be the collection $B = \bigcup_{l=1}^{L-1} b_l$, with $b_l = (b_j^l) \in \mathbb{R}^{n_{l-1}}$ denoting the column vector of biases for layer l such that the objective is to find the optimal $\theta = (W, B)$ parameters. The collections W and B fully determine the output of the entire DNN. Learning is implemented by adapting these weights in order to minimize the error on the training data. In particular, the objective is to minimize some loss function $\mathscr{L}(\theta)$ for each training example $(x_i, y_i) \in D$. Since we are dealing with a classification problem, the loss function is 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 fully parametrized DNN as a function according to W and B of feature vector x_i. Since we are using the softmax function [9] 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 [27].

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

Following [38], [39], and [9], 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 description of

the sequential version of SGD.

Algorithm 4: Stochastic Gradient Descent [9]

```
/* Draw initial parameter values from normal distribution        */
θ ← r with r = (r_i)_{i=1}^n and r_i ∈ N(μ, σ), ∀i
/* Get loss according to initial parametrisation                 */
L ← 0
for i ∈ {1, ..., n} do
    L ← L + L_i(θ)
/* As long as loss above threshold τ, continue to update θ       */
while L ≥ τ do
    for i ∈ {1, ..., n} do
        θ ← θ − λ∇_θ L(θ, i)
    L ← 0
    for i ∈ {1, ..., n} do
        L ← L + L_i(θ)
```

In order to describe the network's topology, we introduce the following notation: 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. A popular rule to set the number of neurons in the first hidden layer H1 of a feedforward network is to use as many neurons as there are inputs [27] which we also abide by. To introduce a closer fit to the data, we also set H1 = H2. Via the third and fourth hidden layer (H3 and H4), we introduce a bottleneck, enforcing a reduction in dimensionality in line with [43] and [9] resulting in H3 = 15 and H4 = 10. Let K be the number of features of the data. Then, our topology specification results in I-H1-H2-H3-O = K-K-K-15-10-2 with input layer I matching the number of features. We train with 400 epochs, i.e., we pass 400 times over the training set, as in [23]. At this stage we would like to point that according to [44] the design of an ANN is more of an art than a science. Therefore, we 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 shows the operational costs aspect of such an endeavour.

4.4 Trading Algorithm

For trading phase, we proceed similarly to [13], [19] and [27]. After having trained each model for different specifications, we generate probability estimates for each class based on the trading sets feature's. These probabilities for each bin are 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 $t + 1$. We refer to these position 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, we enter a long and a short position at most per bin t. In order get a better estimate of the return, we enter 60 initial short and long positions at different points in

time, thus the resulting portfolio has 120 active positions at most. Then, we proceed to calculate aggregate values for each of these positions. To render the backtest more realistic, we incorporate 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, we introduce a execution gap of one minute. This means that when generating the signal from the 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 ← 0
while row_number < (max_row_number − delta) do
    /* Load row data given by row_number                          */
    row ← OHLC.getRow(row_number)
    /* Obtain maximum probability pair max_prob_pair              */
    max_prob_pair ← 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 ← row_number + delta
    else
        row_number ← row_number + 1
```

5 Results

5.1 General Results

After training the models, we evaluated their accuracies on the remaining trading data ranging from 01.11.2019 to 31.12.2019. Before applying the model on the test data, we excluded bins for which no volume was traded in the following bin or lagged values are not available, as was already done to the training set.

	With Volume		No Volume	
Model	**Validation Score**	**Test Score**	**Validation Score**	**Test Score**
Logistic	52.35	53.02	52.18	53.22
Forest	52.71	53.27	52.30	53.53

Figure 4: This table illustrates the accuracy both for the validation and test set for different feature selection.

For the Logistic Regression, we achieved an accuracy of 53.23% and 53.02% with and without volume respectively. The Random Forest achieved an accuracy of 53.54% and 53.28% with and without volume respectively (see figure 4). As one can see for most models, the incorporation of volumes makes no discernable difference. These accuracies are not out of the ordinary when comparing their performance to the results of other methods (see ¡cite Predicting price¿). Due to this and reduced amount of computational effort, we decided to continue with models that were trained without volume.

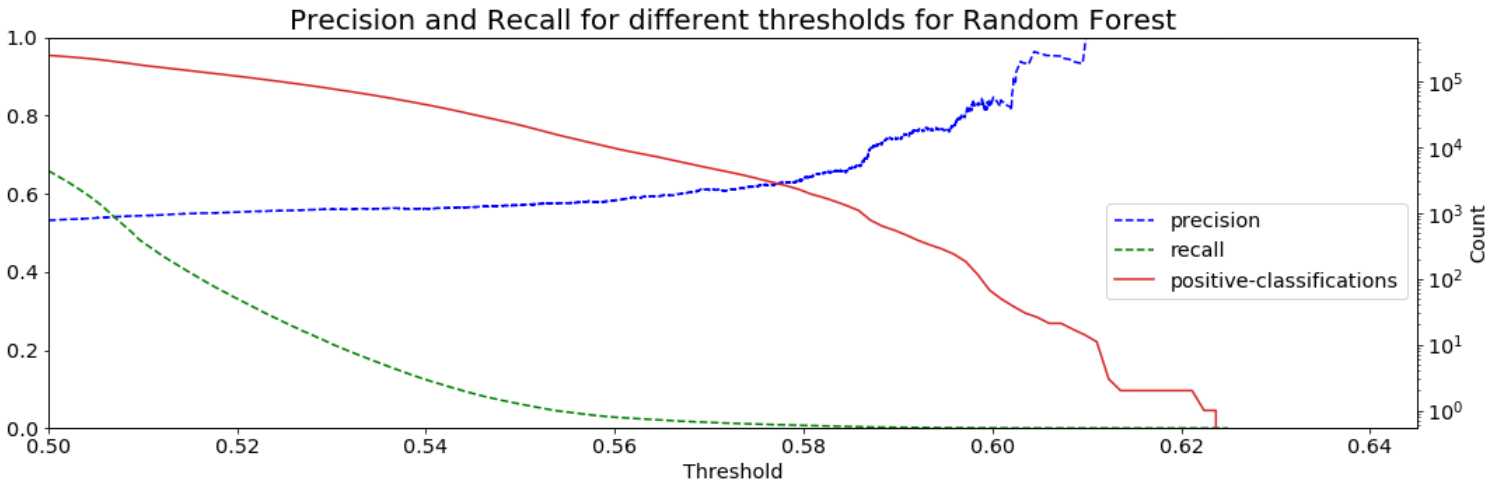

Figure 5: This figure illustrates the precision, recall and up-classifications of the Random Forest for different thresholds without taking volume into consideration.

Figure 5 illustrates the precision, recall and up-classifications of the Random Forest for different thresholds. The slope of the precision of remains relatively stable up until a threshold of 0.58 after which a strict increase takes place for increasing thresholds. On one hand, this comes with the tradeoff that drastically less up-classifications take place, which in turn leads to less trades for the trading algorithm, thus less potential return. On the other hand, strong probability signals indicate relatively high up-movements and low error probabilities, thus higher returns per trade and profitability after transaction cost. Therefore, it is paramount to fine tune the threshold in order to avoid costly missclassification. Only judging from the aforementioned figure, the highest returns will be achieved somewhere around a classification threshold of 0.6, since there are still a considerable amounts of trades in combination with high precision. In chapter 5.2, we demonstrate how changing thresholds leads to drastic differences in terms of return and profitability.

5.2 Strategy Performance

In this chapter, we outline the performance after employing the trading algorithm on the trading set as described in chapter 4.4. Figure 6 illustrates total returns using the Random Forest for probability signals when employing the trading algorithm for different classification threshold levels. For low probability thresholds, the returns are strongly negative, since due to the low precision (see figure 5), a high amount of missclassifications take place. These missclassification then cancel off the positive returns in case of a correct classification.

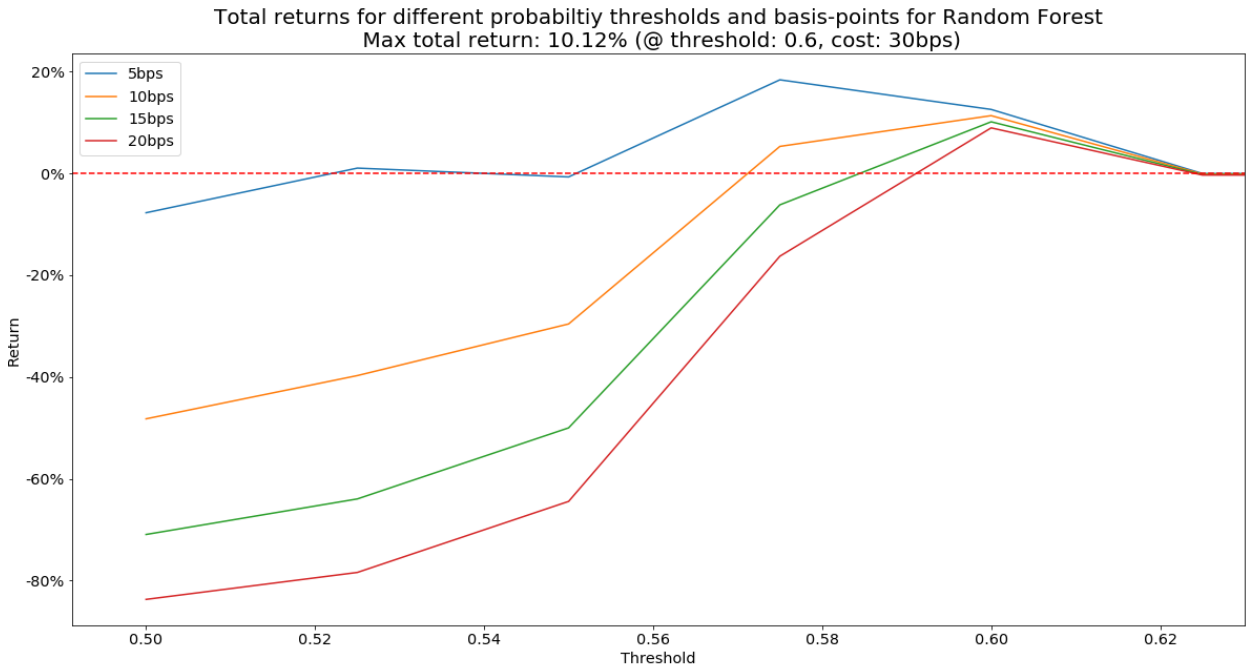

Figure 6: This figure illustrates total returns using the Random Forest for probability signals when employing the trading algorithm for different classification threshold levels. These returns are also further subjected to different levels of transaction cost (10, 20, 30 and 40 bps).

Further, because a weak probability signal is sufficient to surpass the threshold, the proportion of correct classification with insufficient returns is higher, as can be seen in figure 7. Even though the algorithm chose the correct position according to the price movement, the actual realized return can be below the transaction cost, which causes further decline in total return. Therefore, increasing the threshold also increases the proportion of classifications with sufficient return, since these observations with such returns tend to exhibit stronger probabilities.

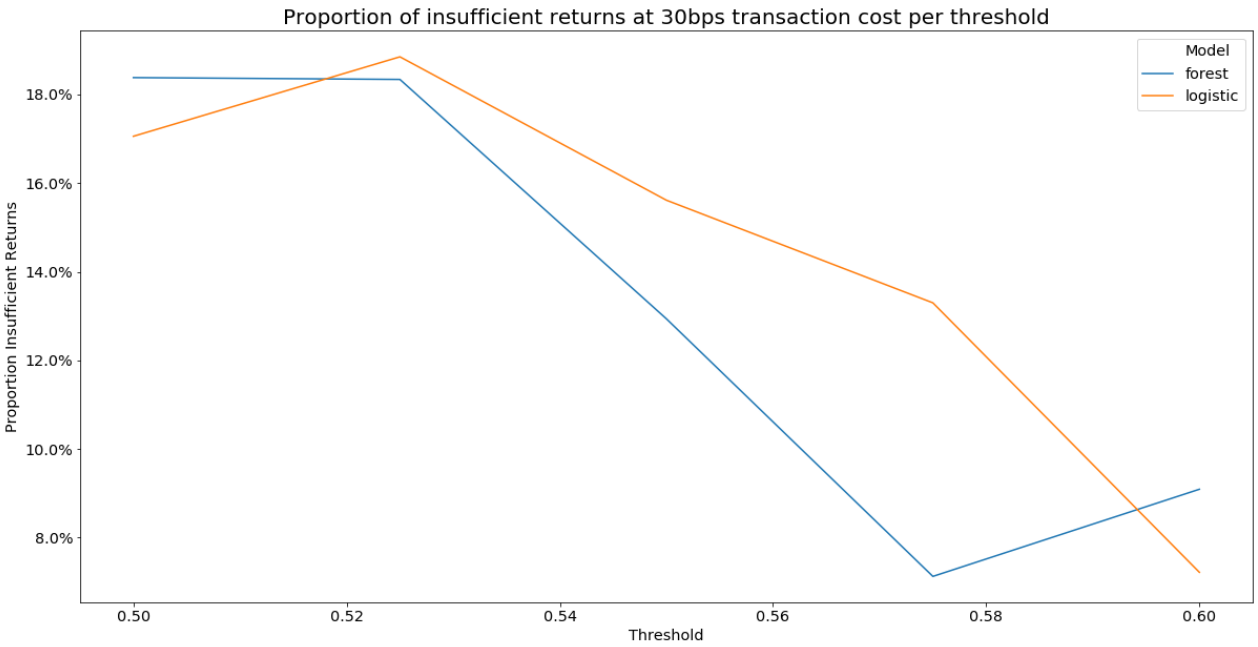

Figure 7: This figure illustrates the total proportion of correct trading decision according to the price movement with insufficient returns at 30bps transaction cost for different thresholds. These returns are insufficient, because they are below the 30bps which leads to effectively negative returns.

For increasing probability threshold, the returns also tend increase up until 9.87% at 30bps for the Random Forest, which can also be explained with the aforementioned implications from figure 5 and 6. Even when introducing different levels of transaction cost, the returns are consistently positive for a threshold of 0.6 converging to similar values. The convergence can be explained by the fact that considerably less trades take place for high threshold values as can be seen in figure 5. Since less trades are conducted, the implicit compound interest effect of transaction cost is drastically reduced, thus slight changes in costs do not affect overall profitability as much. Finally, after a certain threshold level, virtually no positive classifications take place according to figure 5, thus leading to zero returns eventually, which can also be seen for the Logistic Regression in figure 8.

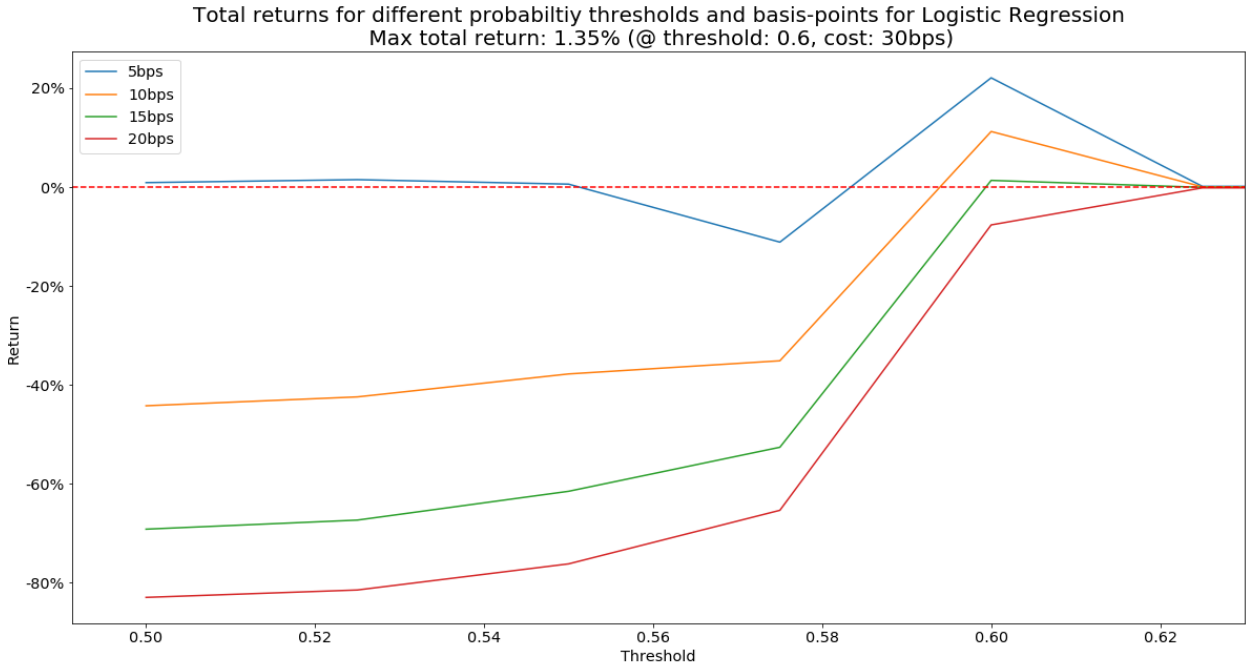

Figure 8: This figure illustrates total returns using the Logistic Regression for probability signals when employing the trading algorithm for different classification threshold levels. These returns are also further subjected to different levels of transaction cost (10, 20, 30 and 40 bps).

5.3 Further Analyses

In this chapter, we analyse how the different return deltas impacted the predictions ofthe model.

Figure 9 illustrates the feature importances for the Random Forest according to MDI [30] for different return deltas in minutes (see chapter 4.2). Surprisingly, the overall shape of of the graph is concave peaking a 60 min instead of peaking right at the beginning and then monotonously decreasing in importance. 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.

Figure 10 illustrates the coefficients for the Logistic Regression for different return deltas in minutes. Comparing the absolute coefficient values of with the MDI values from the Random Forest, the effect of the return deltas seems to be inverted. The highest coefficient values are realized at the beginning and at the end which is in stark contrast to the Random Forest. However, these findings should be viewed critically, since almost all coefficients exhibit insignificant t-statistic values indicated by the blue color.

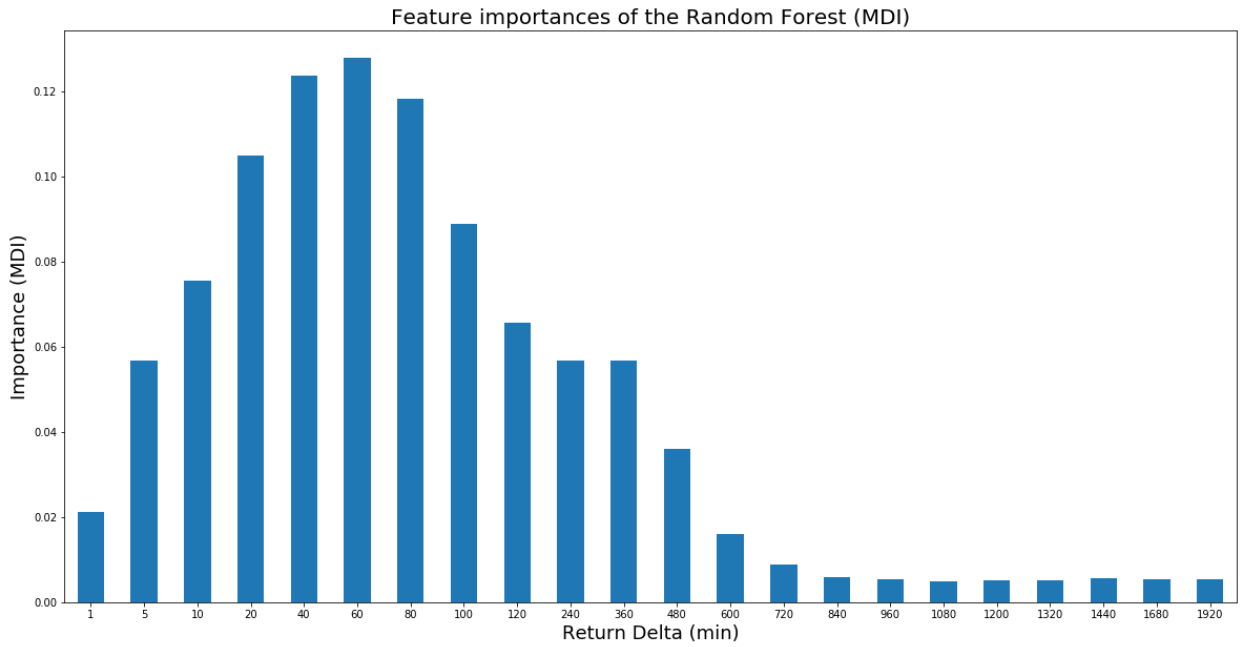

Figure 9: This figure illustrates the feature importances for the Random Forest according to MDI [30] for different return deltas in minutes as described in chapter 4.1.

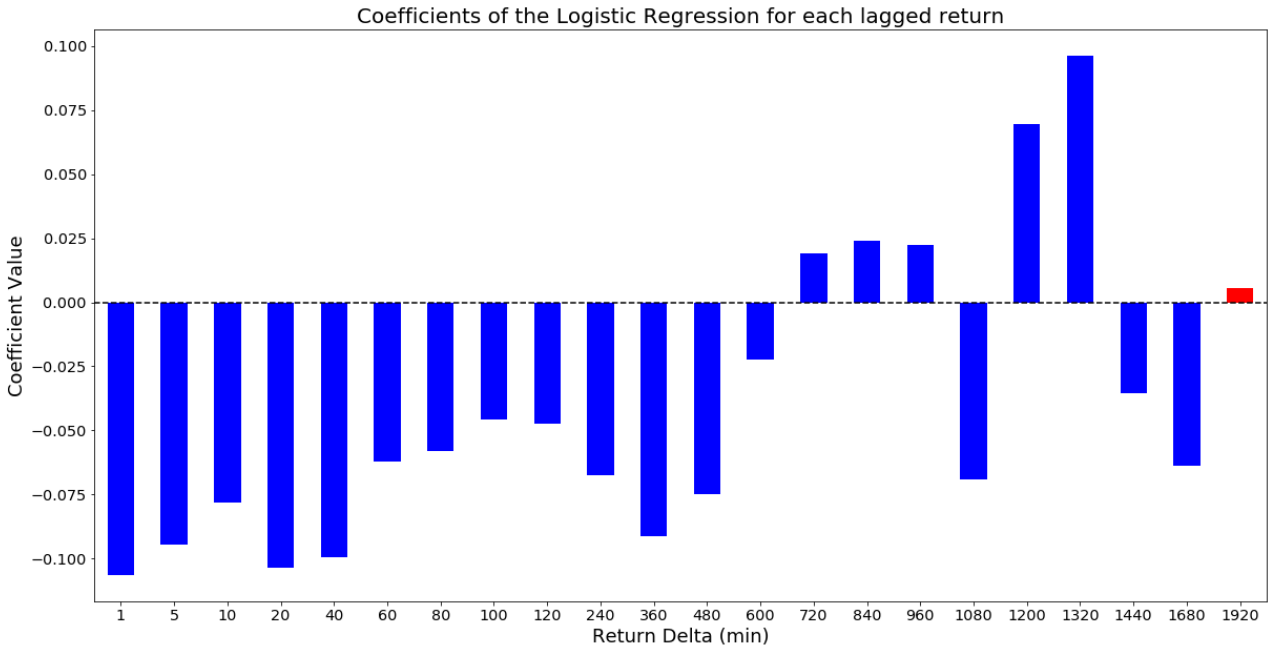

Figure 10: This figure illustrates the coefficients for the Logistic Regression for different return deltas in minutes as described in chapter 4.1. The blue bars indicate non-significant coefficient according to the t-test whereas the red ones are significant.

6 Discussion

When discussing these findings, one has to consider the limits of arbitrage. In our application, they arise mostly from the microstructure of the crypto market. Due to the high trading frequency, the trading strategy is relatively often confronted with liquidity issues. Therefore, in order to facilitate a trade, we followed [19], [4] and [29], and demand that to only trade (i) when volume is present for a coin and (ii) with a one period gap after signal generation. However, when scaling the invested amount in a position, these restrictions are likely not enough for simulating actual trading on the crypto exchange. The greater the order, the longer it takes for the market to clear it. This effect gets even more amplified in markets with low activity and market capitalization. Further, due to the unregulated nature of crypto markets, it is possible to influence markets with large enough orders, especially in markets with low capitalization. This effect is difficult to model, since model also needs to take its own actions into account. A possible solution would be to use an agent based model with profits as reward which could be realized by Deep Reinforcement Learning. Before deploying this method, the microstructure of the market would be rebuild similar to a game. This market would serve as the environment with which the agent of the Deep Reinforcement Learning algortihm interacts with. Another major limit to arbitrage is capacity. An intraday strategy for cryptocurrencies may offer high returns. By contrast, costs for operating such a strategy would be significant, when taking into account human capital and technical infrastructure [13].

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: (2018).

[3] Julius Berner et al. "Towards a regularity theory for ReLU networks – chain rule and global error estimates". In: (2019).

[4] David A. Bowen and Mark C. Hutchinson. "Pairs Trading in the UK Equity Market Risk and Return". In: *European Journal of Finance* (2016).

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

[6] Mark M. Carhart. "On Persistence in Mutual Fund Performance". In: (1997).

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

[8] Thomas G Dietterich. "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization". In: (2000).

[9] Matthew Dixon, Diego Klabjan, and Jin Hoon Bang. "Implementing Deep Neural Networks for Financial Market Prediction on the Intel Xeon Phi". In: (2015).

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

[11] 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).

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

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

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

[15] 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).

[16] Yoav Freund and Robert E. Schapire. "A Short Introduction to Boosting". In: (1996).

[17] Yoav Freund and Robert E. Schapire. "A Short Introduction to Boosting". In: (1999).

[18] Jerome H. Friedman. "Stochastic Gradient Boosting". In: (2000).

[19] 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).

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

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

[22] Robby Houben and Alexander Snyers. "Crypto-assets: Key developments, regulatory concerns and responses". In: (2020).

[23] Nicolas Huck. "Pairs selection and outranking: An application to the S&P 100 index". In: (2009).

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

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

[26] 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).

[27] 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).

[28] 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).

[29] Bo Liu, Lo-Bin Chang, and Helyette Geman. "Intraday Pairs Trading Strategies on High Frequency Data: The Case of Oil Companies". In: *Quantitative Finance* (2015).

[30] 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).

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

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

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

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