Budapesti Corvinus Egyetem

Gazdálkodástudomyányi Kar

Nemzetközi Gazdálkodás Központ

Predicting the S&P 500 with long short-term memory network, gated recurrent unit, and random forest

Készítette: Aczél Till Ariel

Nemzetközi Gazdlkodás szak

2019

Konzulens: Badics Milán Csaba

BUDAPESTI CORVINUS EGYETEM

Gazdálkodástudományi kar

I. számú melléklet

NYILATKOZAT SAJÁT MUNKÁRÓL

Név: ACZÉL TILL ARIEL

Szak: NEMZETKÖZI GAZDÁLKODÁS

A szakdolgozat címe magyarul:

Az S&P 500 ELŐREJELZÉSE LONG SHORT-TERM MEMORY NETWORK, GATED RECURRENT UNIT ÉS RANDOM FOREST HASZNÁLATÁVAL

A szakdolgozat címe angolul:

PREDICTING THE S&P 500 WITH LONG SHORT-TERM MEMORY NETWORK, GATED RECURRENT UNIT, AND RANDOM FOREST

Szakszeminárium-vezető/konzulens neve: BADICS MILÁN CSABA

Én, Aczél Till Ariel (a hallgató neve) **teljes felelősségem tudatában kijelentem, hogy az általam papír alapon és elektronikusan feltöltött szakdolgozat 100%-ban egyezik, és nevezett szakdolgozatban szereplő minden szövegrész, ábra és táblázat – az előírt szabályoknak megfelelően hivatkozott részek kivételével – eredeti és kizárólag a saját munkám eredménye, más dokumentumra vagy közreműködőre nem támaszkodik.**

Kelt: 2019. 10. 16.

hallgató aláírása

Párhuzamos képzés esetén kitöltendő!

Én, ...(a hallgató neve) **teljes felelősségem tudatában kijelentem, hogy a jelen szakdolgozatom és a párhuzamos képzésemen leadott szakdolgozatom közötti átfedés nem haladja meg a 10% százalékot, a Tanulmányi és Vizsgaszabályzat Gazdálkodástudományi kari mellékletének II. fejezetében a 10. § (2) alapján. Tudomásul veszem, hogy amennyiben a szakfelelősök (vagy az általuk megjelölt személyek) 10%-nál nagyobb egyezőséget állapítanak meg, akkor a tanulmányi kötelezettségeimet nem teljesítettem, záróvizsgát nem tehetek.**

Kelt: .....

hallgató aláírása

TÉMAVEZETŐI NYILATKOZAT

Alulírott, BADICS MILÁN CSABA **konzulens kijelentem, hogy a fent megjelölt hallgató fentiek szerinti szakdolgozata (egyetemi/ mesterképzésben diplomamunkája) benyújtásra alkalmas és védésre ajánlom.**

Budapest, 2019. 10. 16.

Badics Milán Csaba

konzulens aláírása

Bekötés helye: A Szakdolgozat belső címlapját követő oldal (a „Tartalomjegyzék-et megelőzően).

BUDAPESTI CORVINUS EGYETEM
Gazdálkodástudományi kar

II. számú melléklet

NYILATKOZAT A SZAKDOLGOZAT NYILVÁNOSSÁGÁRÓL

Név (nyomtatott betűvel): ACZÉL TILL ARIEL

Szak neve (amelyen a jelenlegi Szakdolgozatát megírta): NEMZETKÖZI GAZDÁLKODÁS

Dolgozatom elektronikus változatának (pdf dokumentum, a megtekintés, a mentés és a nyomtatás engedélyezett, szerkesztés nem) nyilvánosságáról az alábbi lehetőségek közül kiválasztott hozzáférési szabályzat szerint rendelkezem.

TELJES NYILVÁNOSSÁGGAL

A könyvtári honlapon keresztül elérhető a SPzakdolgozatok/TDK adatbázisban (http://szd.lib.uni-corvinus.hu/), a világháló bármely pontjáról hozzáférhető, fentebb jellemzett pdf dokumentum formájában.

KORLÁTOZOTT NYILVÁNOSSÁGGAL

A könyvtári honlapon keresztül elérhető a Szakdolgozatok/TDK adatbázisban (http://szd.lib.uni-corvinus.hu/), a kizárólag a Budapesti Corvinus Egyetem területéről hozzáférhető, fentebb jellemzett pdf dokumentum formájában.

NEM NYILVÁNOS

A dolgozat a BCE Központi Könyvtárának nyilvántartásában semmilyen formában (bibliográfiai leírás vagy teljes szöveges változat) nem szerepel.

Budapest, 2019.10.14.

hallgató (szerző) aláírása

Acknowledgements

I would first thank my thesis supervisor, Milán Badics for the continuous support. Without him, it would have been much harder to get into the field of machine learning. I am grateful for his guidance at the beginning of my research, for the lectures he gave at the machine learning research group, and for his support and constructive criticism during the writing of this thesis.

Besides my supervisor, I would like to thank Tulián Aczél for providing the necessary hardware to train the models. My sincere thanks also goes to Leandro Arenas Arce and Máté Uzsoki, for their insightful comments. Last but not least, I would like to thank my parents for supporting me throughout my BSc and my life in general.

Abstract

Long short-term memory (LSTM) and gated recurrent unit (GRU) are state-of-the-art sequence modelling techniques, while random forest (RF) is a traditional machine learning model. The autoregressive moving average (ARMA) model is a well known econometric model, used for financial sequence predicting for decades. In this paper, I am looking at the predicting power of these models and the profitability of a trading strategy based on the prediction of these models at different trading frequencies. In most of the financial time series prediction papers, the prediction and trading happen at a daily frequency. I examine the one time period ahead forecasting of these models from a minute by minute to a daily frequency. The models are trained, validated and tested on the S&P 500 index from 2008 to 2017.

Contents

List of Figures

List of Tables

1 Introduction

Algorithmic trading exists since computers exist. It was an obvious idea to use the new computational power and econometric models to predict the stock market and speculate on it. The effective market hypothesis (Malkiel and Fama 1970) states that market prices fully reflect all available information. This is only true to the degree to which the information can be analyzed and traded upon it. The trading based on computers and econometric models made those models inefficient, because to the degree those models could process the information of past prices was already present in the current prices of the assets. With better models, the market got more efficient. The predictive models are an arms race between hedge funds.

Although deep learning only got much attention since 2012 hedge funds are using them since the beginning of the century (Krauss, Do, and Huck 2017). Aldridge and Krawciw (2017) estimated that high-frequency trading accounted for 10–40% of trading volume in equities. Since 2017 the importance of high-frequency trading increased even more. Because high frequency trading requires electronic trading, the amount of financial data increased exponentially. The explosion of computing power and data enabled the use of machine learning models. Because of the value of the newest models, there is a gap between the academic literature and the industry.

In this thesis I attempt to bridge this gap by looking at the effects of the frequency of the trading on the profitability of the long short-term memory (LSTM), gated recurrent unit models (GRU), random forest (RF) and an equal weighted ensemble (ENS) against a benchmark of an autoregressive moving average model (ARMA). Most research papers compare machine learning and econometric models on daily frequency. I am examining the S&P 500 index prices from 2008 to 2017 from minute by minute to daily frequency. For reproductivity purposes, I run the models also on publicly available Bitcoin data.

The remainder of this thesis is organised as follows. Section 2 outlines the relevant literature and my contribution to the field. Section 3 provides a summary of the theoretical background of the models used in the thesis. Section 4 covers the methodology. Section 5 presents the results and key findings. Finally, section 6 concludes.

2 Literature review

Krauss, Do, and Huck (2017) compared deep neural networks, gradient-boosted trees, and random forest one day ahead directional predicting capabilities. They used lagged returns from 1992 to 2015 of all the S&P 500 stocks - after eliminating survival bias - to predict the directional movement of each stock. They took a long position in the k best predicted stocks and a short in the k worst predicted stocks. Their results showed that random forest outperformed gradient-boosted trees and deep neural networks. Krauss, Do, and Huck (2017) used a rule of thumb method for the hyperparameters of the neural network, so its performance can most likely be increased. Their findings showed, that an equal weighted ensemble of these models outperforms all of the individual models. The equal weighted ensemble with $k = 10$ achieved a 73% annualized return with a sharp ratio of 1.81 which is roughly five times higher than the general market. From 2001 the returns of the trading strategies start to diminish. They assumed this is caused by the deployment of these strategies, which leads to the returns being arbitraged away and to higher market efficiency. During highly volatile time periods (global financial crisis) all strategies perform well. Krauss, Do, and Huck (2017) also showed that on a daily frequency the most recent days have the highest importance in predicting the next day's directional movement.

Fischer and Krauss (2018) extended Krauss, Do, and Huck (2017) work by comparing long short-term memory networks to random forest, deep neural networks, and logistic regression classifiers. They used the same data, study periods, trading strategy and evaluation as Krauss, Do, and Huck (2017). Their findings show that the long short-term memory network outperforms random forest, deep neural networks, and logistic regression classifiers on this dataset. The long short-term memory network is able to achieve a 0.26% daily return after transaction cost and a sharp ratio of 2.34. The returns of the long short term memory network diminish after 2010. Fischer and Krauss (2018) found that the long short-term memory portfolio consists of stocks with below-mean momentum, strong short-term reversal characteristics, high volatility and beta. Based on these common patterns, they could derive a rule-based trading strategy, which achieved a daily return of 0.23% prior to transactional costs.

Alonso, Batres-Estrada, and Moulin (2018) selected 50 stocks from the S&P 500 index. In their first experiment, they used the past returns of one stock up to time t as input to predict the return at time $t + 1$ of the same stock, where Δt is one day. The long

short-term memory network predicted the direction of the price movement correctly over 50% of the time for 43 stocks out of 50 stocks, the support vector machine for 21 stocks and the feed forward neural network for 27 stocks. In their second experiment, they used all of the 50 stocks returns to predict the return of one stock at time $t + 1$. The long short-term memory network improved its performance compared to the first experiment.

Shen et al. (2018) deployed the gated recurrent unit network to a feed forward neural network, a support vector machine and a combination of the gated recurrent unit and the support vector machine on the HIS, DAX and S&P 500. The inputs of the models were the past 240 daily returns of the indexes. Shen et al. (2018) composed 23 study periods, with 750 samples in the training set and 250 samples in the trading set. On all of the three indexes, the gated recurrent unit network has higher accuracy on the trading set. On the S&P 500, the standard gated recurrent unit network predicted the direction of the price movement 51.4% of the time, while the combined model 51.7% of the time.

Borovkova and Tsiamas (2018) compared several ensembles of long short-term memory networks, with lasso and ridge regression for intraday directional forecast. They selected 44 large and liquid stock from the US market and tested their predictive power on 22 stocks. Borovkova and Tsiamas (2018) deployed these models on one year (2014) of five minute interval data. They generated features for the long short-term memory network. These features included open, close, low, high, volume, standard deviation and more. Borovkova and Tsiamas (2018) introduced layer normalization to the long short-term memory network to prevent neurons from saturating and dropout to prevent it from overfitting. For the long short-term neural network, the problem was faced as a classification problem. The ensemble models were equal weighted ensemble, performance weighted ensemble, and best model ensemble. The ensembles consisted of 12 long short-term memory networks with randomized input features, to achieve the required diversity of the models for the ensemble to work. They used 21 study periods, with roughly one month training, one week validation and one week trading. Evaluation of the performance was done by the area under the ROC curve score. The performance weighted ensemble produced significantly better results than the other methods.

I am expanding this literature by focusing on the effect of the trading frequency on the performance of state-of-the-art machine learning methods. Furthermore, I explore how the predicting capabilities of the models change at different frequencies. Predictions are useless if it is not possible to construct a profitable trading strategy based on those predictions. I deploy a trading strategy based on the predictions of the models and study

how their logarithmic return changes by introducing market frictions. In short, I want to answer the following questions:

- Does the ARMA, LSTM, GRU, RF or ENS outperform the other models in predictive capabilities or trading strategy performance?
- How does the frequency affect the predictive capabilities and the trading strategy performance?
- How does market efficiency change over time?

3 Models

3.1 Machine learning concepts

The error function (also referred to as cost function, or loss function) defines the cost of the deviation between the output vector $\hat{y}$, and the label vector y. In other words, the error function defines the cost of being wrong by a certain amount. Commonly used error functions for regression are:

- Mean squared error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 \tag{3.1}$$

- Mean absolute error (MAE)

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y^{(i)} - \hat{y}^{(i)}| \tag{3.2}$$

These error functions are used in econometric models as well as in machine learning models.

While building a predictor the bias-variance problem needs to be considered. Bias is an error from not being able to learn complex enough mapping. Variance is an error from learning the noise in the data. Bias and variance need to be minimalized simultaneously so that the model can make the correct input-output mapping beyond the training set - see figure 3.1 (Haykin et al. 2009). The bias-variance tradeoff needs to be considered more in machine learning models. If a model has high bias it underfits the training data, if it has high variance it overfits it.

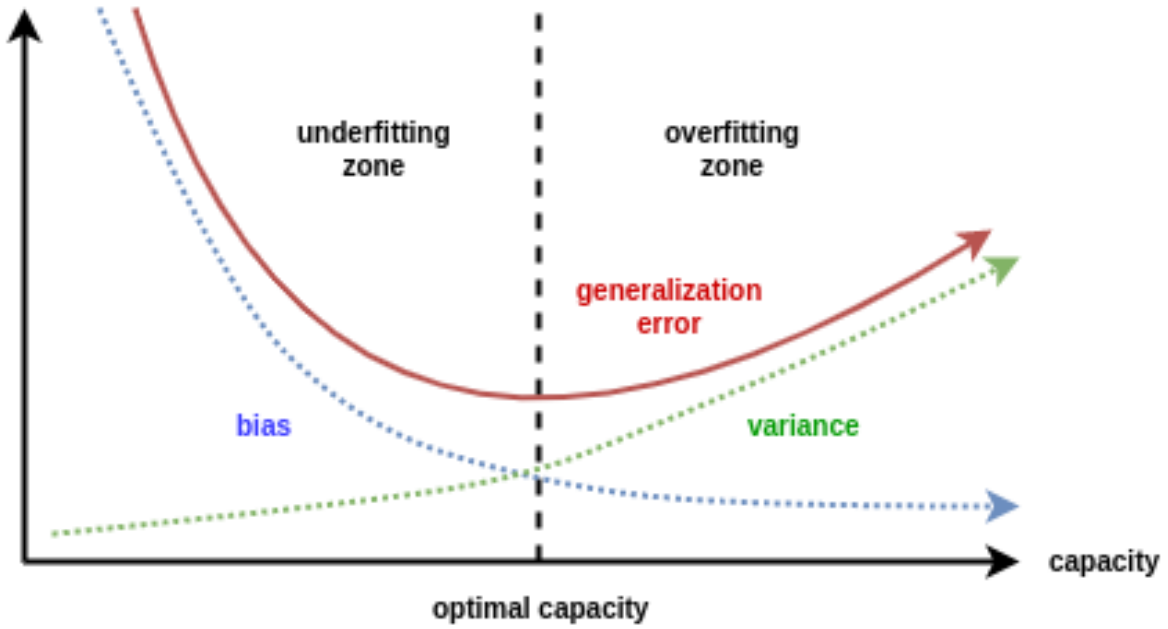


Figure 3.1: Bias-variance tradeoff as a function of model capacity - (Saunders 2017)

Hyperparameters are the parameters that can not be learned during training. These parameters need to be chosen before the training. If the hyperparameters are not set optimally then the model will perform badly. The ARMA model has only 2 discrete hyperparameters (p, q), while a neural network has several discrete and continuous hyperparameters. Human experts are bad in hyperparameter estimation, so usually, it is done by trial and error or by a hyperparameter optimalization algorithm. To prevent overfitting, different hyperparameter settings can be compared by the validation set error, or with an information criteria like the bayesian information criterion (Schwarz et al. 1978) or the Akaike's information criterion (Akaike 1998). The ARMA model hyperparameters are usually chosen with an information criterion, while machine learning models with more parameters are validated on the validation set. To get more reliable results, a hyperparameter optimization algorithm can be used. Some of these algorithms are random search, grid search, and bayesian optimization. Random search is a naive approach where the hyperparameters are chosen randomly from a uniform distribution. Grid search is a more systematic search of the hyperparameters. In grid search, a vector of values is defined for each hyperparameter and the validation error is calculated for each combination. Bayesian hyperparameter optimization (Feurer and Hutter 2019) is a stepwise solution, where the hyperparameter search is considered as a supervised learning problem. Before choosing a new set of hyperparameters, it fits a bayesian process on the hyperparameter-validation error pairs and picks the next set of hyperparameters, where

the expected decrease of the validation error is the highest.

In random search, grid search and as well in bayesian hyperparameter optimization the range in which the hyperparameters are explored needs to be defined. If the range is not defined optimally, then a suboptimal hyperparameter setting will be chosen. Usually, random search is preferred over grid search, because if the performance of the model is really sensitive to one of the parameters the random search can explore more different values of that parameter (Bergstra and Bengio 2012). To get the benefit of grid search and random search an option is to combine the two methods, by adding random noise to grid search. Bayesian optimization is preferred over both random and grid search (Feurer and Hutter 2019).

Each hyperparameter gives a dimension in which the hyperparameters needs to be optimized, so each of them increases the complexity of the problem exponentially. Because of that, it is important to compare the hyperparameters of the different models. An ARMA model has only two hyperparameters (p, q). In a feed forward neural network some of the hyperparamteres are: initialization, number of hidden layers, number of nodes per hidden layer, activation functions in each hidden layer, error function, optimizer, learning rate, number of epochs, batch size and if regularization is used, then the regularization is as well. As you can see there are many more hyperparameters to be optimized in a neural network, so it is a much more challenging task.

3.2 The Autoregressive moving average model

A commonly used model for financial time series forecasting is the autoregressive moving average model (ARMA). ARMA models are widely used because they are flexible, simple to estimate, can often produce reasonable forecasts and because they require only the past observations of the time series (Brooks 2008). An ARMA(p, q) process where $q = 0$ reduces to an autoregressive process of order p AR(p) and an ARMA(p, q) process where $p = 0$ to a moving average process of order q MA(q) (Neusser et al. 2016). The ARMA process is weakly stationary, which means it has a constant mean and variance over time. Returns can be validly considered to be stationary (Brooks 2008).

AR: Autoregression. A linear regression model that uses the dependencies between an observation and a number of lagged observations (p) (Siami-Namini and Namin 2018).

An AR(p) process can be defined with equation 3.3,

$$x_t = c + \sum_{i=1}^{p} \phi_i x_{t-i} + \varepsilon_t \tag{3.3}$$

where x_t is the stationary variable, c is a constant, the terms ϕ are model parameters and the residuals, $\varepsilon \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$ are Gaussian white noise (Brockwell, Davis, and Fienberg 1991).

MA: Moving Average. A model that takes into account the dependency between observations and the residual error terms of the model at previous lagged observations (q) (Siami-Namini and Namin 2018). A MA(q) process can be defined with equation 3.4,

$$x_t = \mu + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t \tag{3.4}$$

where μ is the expectation of x_t (usually assumed equal to zero), the θ terms are the weights applied to the prior values of a stochastic term in the time series, and the residuals, $\varepsilon \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$, are Gaussian white noise (Brockwell, Davis, and Fienberg 1991)

ARMA: The AR and the MA process can be combined in to the ARMA model by adding them together - see equation 3.5,

$$x_t = c + \sum_{i=1}^{p} \phi_i x_{t-1} + \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-1} \tag{3.5}$$

where $\phi_i \neq 0$, $\theta_i \neq 0$, and $\sigma_\varepsilon^2 > 0$. The μ is incorporated into the constant term. The parameters p and q are called the AR and MA orders, respectively.

The order of the ARMA process is usually identified using the Box-Jenkins method (Box et al. 2015). The Box-Jenkins method consists of the following steps: stationarity, seasonality, order of ARMA model, estimating the model's parameters, diagnostic checks (Makridakis and Hibon 1997). To visually understand the order, it is wise to look at the autocorrelation function (ACF) and partial autocorrelation function (PACF). An AR process with order p has a PACF with significant values at the first p lags and an MA process with order q has an ACF with significant values at the first q lags (Hyndman and Athanasopoulos 2018).

3.3 Multilayer perceptron

Before recurrent neural networks it can be useful to understand the multilayer perceptron (MLP). MLP is a feedforward neural network with an input layer, one or more hidden layers and an output layer. Each hidden layer, and the output layer consists of perceptrons

(also referd to as artificial neurons), and their input is the output of the previous layer. An artificial neuron is a non-linear transformation of the linear combination of its inputs, see equation 3.6, where a_j^i is the output from the j^{th} neuron in the i^{th} layer, g is the activation function, w_j^i and b_j^i are the weight vector and bias from the j^{th} neuron in the i^{th} layer and a^{i-1} is the output vector of the $(i-1)^{th}$ layer. A neuron with a linear activation function is a linear regression.

$$a_j^i = g(w_j^i a^{i-1} + b_j^i) \tag{3.6}$$

$$a^i = g(W^i a^{i-1} + b^i) \tag{3.7}$$

The weight vector has the same length as the input vector. The input of the activation function $w_j^i a^{i-1}$ may also be referred to as z_j^i. To compute one layer at the same time the weight vectors and biases of the layer can be concatenated together in a W^i matrix and b^i vector - see equation 3.7.

The universal approximation theorem states (Hornik, Stinchcombe, and White 1989), that feed forward neural network with a single hidden layer with finite neurons is a universal approximator, so a MLP can approximate any function as precisely as wished. This is a big advantage of neural networks because the type of nonlinearity does not need to be defined in advance. Although one hidden layer is enough, a network with fewer parameters, but more hidden layers - so fewer perceptrons per layer - is able to approximate the underlying function as good as the network with one layer and more parameters, so it is less prone to overfit the data (Lin, Tegmark, and Rolnick 2017).

3.3.1 Activation functions

Activation functions give nonlienarity to the neural networks. Activation functions must be differentiable (Haykin et al. 2009) (some activation functions are not differentiable in some points, in those cases it can be assumed, that the derivative of the activation function returns one side of the one-sided derivatives). Some of the most commonly used activation functions are logistic sigmoid (equation 3.8), tangent hyperbolic (TanH, equation 3.9) and rectified linear unit (ReLU, equation 3.10) - see figure 3.2.

$$\text{Sigmoid: } g(x) = \frac{1}{1+e^{-x}} \tag{3.8}$$

$$\text{TanH: } g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.9}$$

$$\text{ReLU: } g(x) = \max(0, x) \tag{3.10}$$

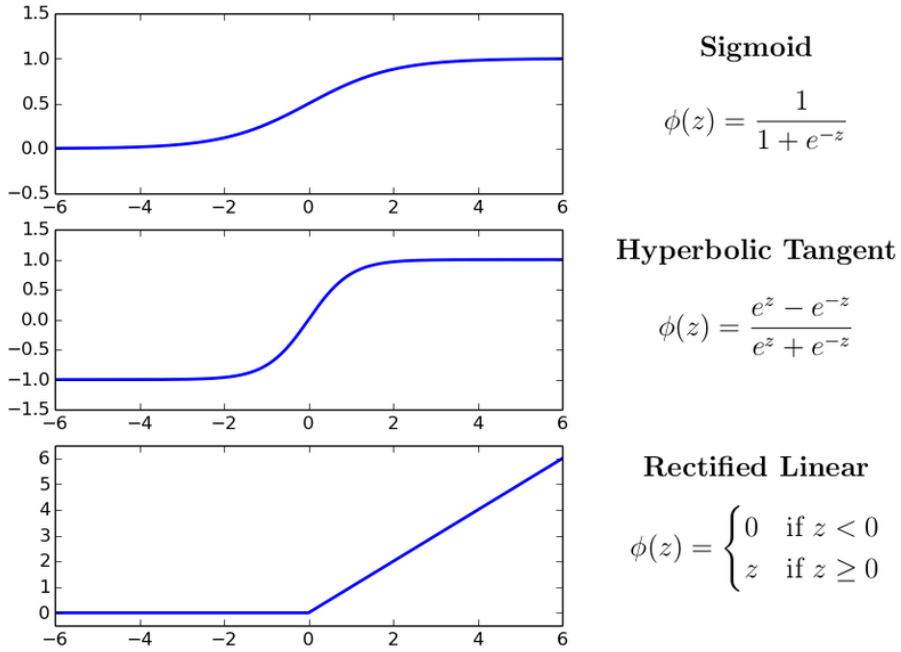


Figure 3.2: Commonly used activation functions - (Hughes and Correll 2016)

3.3.2 Bias and variance in neural networks

Deep neural networks are powerful, but because of the large number of parameters, they are prone to overfitting. This is especially true if the training dataset is small (Srivastava et al. 2014). One option to measure overfitting is to split the data into training, validation and test set. The network is trained on the training set, the hyperparameters are optimized on the validation set and the result is evaluated on the test set. Training set should be around 40-80 percent, validation 10-30 percent and test 10-30 percent. It is important that there is no information about the validation set or the test set in the training set, and that there is no information about the test set in the validation set. Time series have temporal dependencies, "'peeking' at the held-out data while selecting the forecasting method pollutes the evaluation environment" (Tashman 2000), so the data needs to be split temporally. To divide the data into training, validation, and test set requires a large amount of data, thus this method may not be applicable for problems where data is limited (Zhang, Patuwo, and Hu 2001).

Regularization aims to reduce the test error by reducing variance, sometimes at the expense of increased training error by increasing bias (Goodfellow et al. 2016). Because deep learning models have a lot more hyperparameters than econometric models they are a lot more prone to overfit the data. Dropout is a regularization technic, that addresses the problem of overfitting, by "thinning" the network. Dropout removes stochastically

units from the network by applying a random binary mask to the hidden layers. The units stay in the network with a given probability. This probability can be different for each layer and it is another hyperparameter of the network. For each forward and backward pass, some of the units are multiplied by 0 giving the effect of dropping them out. By removing units stochastically they can not relay that much on each other, so they need to be more generally useful (Srivastava et al. 2014).

3.3.3 Backpropagation training algorithm

A MLP is a deeply nested function with parameters (weights). Backpropagation (BP) training algorithm (Rumelhart, Hinton, and Williams 1986) updates the weights with the help of the partial derivatives (gradients) in respect to the weights, so that the neural network makes a correct input to output space mapping. BP has the following steps:

Algorithm 1 Backpropagation

1: Weight initialisation
2: **while** stopping criteria is not met **do**
3: Propagating forward the data to calculating the output ($\hat{y}$) ▷ Forward pass
4: Calculating the cost from the deviation
5: Calculating the gradients with respect to each weight ▷ Backward pass
6: Updating the weights

In the standard BP algorithm the weights are updated each iteration according to equation 3.11, where α is the learning rate, E is the error function and $w^i_{j,k}$ is the k^{th} weight from the j^{th} neuron in the i^{th} layer (Altrichter et al. 2006).

$$\Delta w^i_{j,k} = -\alpha \frac{\partial E}{\partial w^i_{j,k}} \tag{3.11}$$

Before training, the value of the learning rate α has to be set. α defines the rate with which the weights get updated. It is important to set the learning rate correctly. "A large learning rate may cause the problem of oscillation. To overcome this problem a small learning rate is recommended, however a small learning rate may get the network stuck at a local minimum before learning the whole training set" (Jin et al. 2000). The smaller the learning rate the slower converges the network to the minimum point. The zeroth step of BP is to initialize the weights. It is important to give different starting values to the weights to break up the symmetry. It is also important that the weights get

initialized around 1, because of the exploding, and vanishing gradient problem. You can further read about the exploding and vanishing gradient problem in section 3.3.4.

Feature scaling accelerates the convergence of the ANN, because the weights do not have to take on extreme values. Some of the scalings are min-max scaling - see equation 3.12, and standardization - see equation 3.13, where σ is the standard deviation of the dataset x.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.12}$$

$$x' = \frac{x - \text{average}(x)}{\sigma} \tag{3.13}$$

The batch size determines how many forward and backward passes are made before the error is calculated and the weights are updated. An epoch is when the entire training dataset is used once to train the network. Batch learning is when an epoch consists of one batch, which means the batch size is equal to the training set size, so the error is calculated once and the weights are updated once per epoch. In online learning, the weights are updated example-by-example, which means batch size is equal to 1, so the error is calculated for each data sample separately and the weights are updated each error calculation (Haykin et al. 2009). Stochastic gradient descent means that the batch size is between 1 and the training set size. The smaller the batch size the computationally faster the backpropagation training algorithm (Haykin et al. 2009). The stochasticity of the training algorithm has the desirable effect that it is less prone to getting stuck in a local minimum of the error surface (Haykin et al. 2009), but the model can not fit the whole training dataset as accurately as by the batch learning.

3.3.4 Vanishing gradient problem

Vanishing gradient problem was discovered by Hochreiter in 1991. When calculating the gradient the chain rule has to be applied, which means, that the derivatives of the activation function and the weights of the previous layers get multiplied. If those values are not around 1 the gradient either explodes or vanishes. The derivative of the logistic sigmoid function has a codomain of $[0, 0.25]$, so neural networks with the logistic sigmoid activation function struggle to train the first layers because the gradient decreases exponentially the further down the backward pass goes. In deep neural networks, this problem gets amplified because they have a lot of layers.

3.4 Recurrent neural network

A recurrent neural network (RNN) is an extension of a feedforward neural network, for serial data. One of the RNN-s inputs is the output calculated at the previous data point in the sequence. A RNN has a loop in it, which allows information (hidden state) to pass from one time step to the next time step. At each time step, the output vector is calculated by applying an activation function to the hidden state vector. In this section, I am writing about the vanilla RNN and about two modifications of it: the long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) and the gated recurrent unit (GRU) (Cho et al. 2014), which gained attention in recent years. These layers in a RNN define the mapping between the hidden state of the previous time step, the new input and the hidden state of the current time step. The hidden state for the first time step is usually set to 0 (Chung et al. 2014).

3.4.1 Vanilla recurrent neural network

A cell of a vanilla RNN is defined in equation 3.14, where h_t is the hidden state vector of time step t, σ_h is the tanh activation function, h_{t-1} is the hidden state vector of the previous time step, W and b are the weight matrix and bias, and x_t is the input vector (Lipton, Berkowitz, and Elkan 2015).

$$h_t = \sigma_h(W[h_{t-1}, x_t] + b) \tag{3.14}$$

A RNN is not so different from a standard feed forward neural network. A RNN can be unrolled by copying the cells beside each other.

It has been presented, that vanilla RNN often outperform static models (Bengio 1991) howewer, theoretical and experimental evidence shows that a vanilla RNN is not optimal for tasks involving long-term dependencies (Bengio, Simard, and Frasconi 1994).

3.4.2 Long short-term memory

Long short-term memory (LSTM) was proposed in 1997 by Hochreiter and Schmidhuber. A cell of the LSTM can be defined by the following equations:

$$f_t = \sigma_g(W_f[h_{t-1}, x_t] + b_f) \tag{3.15}$$

$$i_t = \sigma_g(W_i[h_{t-1}, x_t] + b_i) \tag{3.16}$$

$$o_t = \sigma_g(W_o[h_{t-1}, x_t] + b_o) \tag{3.17}$$

$$\tilde{C}_t = \sigma_h(W_C[h_{t-1}, x_t] + b_C) \tag{3.18}$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \tag{3.19}$$

$$h_t = o_t \circ \sigma_h(C_t) \tag{3.20}$$

where f_t is the forget gate vector, i_t is the input gate vector, o_t is the output gate vector, σ_g is the logistic sigmoid function, σ_h is the tanh activation function, h_{t-1} is the hidden state vector of the previous time step, x_t is the input vector, W_f, W_i, W_o and W_C are the weight matrices, b_f, b_i, b_o and b_C are the bias vectors, $\tilde{C}_t$ is the candidate value vector, C_t is the cell state vector, h_t is the hidden state vector of the current time step and $\circ$ is the hadamard product (Jozefowicz, Zaremba, and Sutskever 2015). You can see a visual representation of a LSTM in figure 3.3.

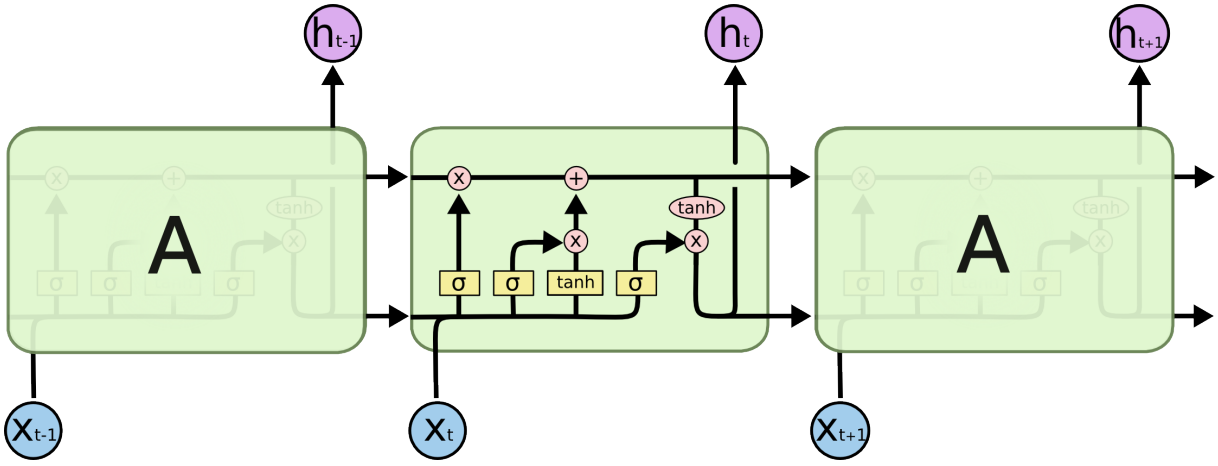


Figure 3.3: A layer of a RNN with LSTM cell - (Olah 2015)

You can think of C_t as the memory of the cell. This memory is updated each time step using the three gates. The gates are a feed forward neural network with no hidden layer. The forget gate decides what to forget from the cell state, the input gate decides what should be stored from the candidate value in the cell state and the output gate decides what to forward from the cell state to the hidden state. Unlike the vanilla RNN which overwrites its hidden state each time step, a LSTM cell is able to decide what to keep and what to update in the memory (Chung et al. 2014). The LSTM addresses the vanishing gradient problem (section 3.3.4) by introducing memory and gates (Sutskever 2013).

3.4.3 Gated recurrent unit

Gated recurrent unit (GRU) was introduced by Cho et al. (2014). "Similarly to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit,

however, without having a separate memory cell" (Chung et al. 2014). A GRU can be defined by the following equations:

$$z_t = \sigma_g(W_z[h_{t-1}, x_t] + b_z) \tag{3.21}$$

$$r_t = \sigma_g(W_r[h_{t-1}, x_t] + b_r) \tag{3.22}$$

$$\tilde{h}_t = \sigma_h(W_h[r_t \circ h_{t-1}, x_t] + b_h) \tag{3.23}$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t \tag{3.24}$$

where z_t is the update gate vector, r_t is the reset gate vector, σ_g is the logistic sigmoid function, σ_h is the tanh activation function, h_{t-1} is the hidden state vector of the previous time step, x_t is the input vector, W_z, W_r and W_h are the weight matrices, b_z, b_r and b_h are the bias vectors, $\tilde{h}_t$ is the candidate value vector, h_t is the hidden state vector of the current time step and $\circ$ is the hadamard product (Jozefowicz, Zaremba, and Sutskever 2015). You can see a visual representation of a GRU cell in figure 3.4.

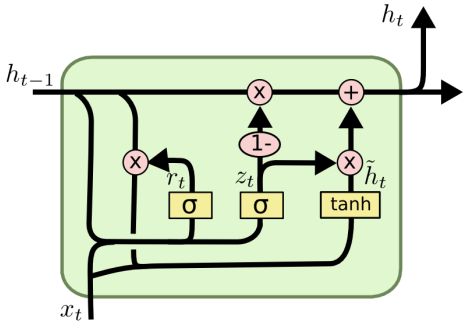


Figure 3.4: A GRU cell - (Olah 2015)

The GRU is a big variation of the LSTM. A GRU has only two gates instead of three and no memory, so it has fewer weights than a LSTM and takes less time to train (Chung et al. 2014). Fewer weights mean that a GRU needs less data because it is less prone to overfitting. In financial time series, this could be an advantage because even high-frequency data like minute by minute data has less than 400 data points per day. One disadvantage of the GRU is, that it is less capable than a LSTM to learn long time dependencies. A GRU, unlike the LSTM, is unable to control what to expose from the information learned in the previous time steps (Chung et al. 2014).

3.4.4 Backpropagation training algorithm trough time

Backpropagation trough time (BPTT) was proposed by Elman (1990). "The backpropagation through time approach can be derived by unfolding the temporal operation of a

network into a multilayer feedforward network that grows by one layer on each time step" (Williams and Peng 1990). Mathematically BPTT is the same as BP if the recurrent network is unrolled, the only difference is that the weights occur multiple times in the network. Every time step can be interpreted as a layer in the network. If the sequence of data on which the model is trained is long, the GPU needs to store a lot of data. This means, that because GPU memory is limited BPTT becomes computationally expensive as the number of time steps is increasing, which slows down the training process (Sutskever 2013). Vanishing gradient problem does not only occur in deep neural networks, but also in RNN-s. A solution for the vanishing gradient problem and the limited GPU memory is truncated backpropagation trough time (TBPTT). TBPTT sets a maximum number of time steps for which the gradient is calculated. One drawback of TBPTT is that it is not capable of learning long term time dependencies (Lipton, Berkowitz, and Elkan 2015).

3.5 Random forest

Random forest (RF) is an ensemble model consisting of decision trees (Svetnik et al. 2003). First I am outlining how a decision tree is constructed. Second I describe how one of the ensemble methods, random forest works.

3.5.1 Decision tree

A decision tree is a nonparametric machine learning algorithm used for classification and regression. The Classification And Regression Tree (CART) algorithm ask multiple yes-no question after each other. These yes-no questions are called nodes. The first of these questions is the root node. The leaf nodes are the output of the model, which can either be a classification label or a regression value. In a decision tree to predict the output for an input, we have to start from the root node. At each node, we follow the branch determined by this yes-no question. We follow the branches until we get to a leaf node (Lewis 2000).

The CART algorithm goes the following. First, the splitting function for the root node needs to be determined. To determine which feature should be used at the node's splitting function, a splitting criteria, like the gini coefficient is calculated (Lewis 2000). For categorical features, the splitting function is calculated for each category, for continuous features the data is ordered by that feature and the average of each adjacent entry is considered. The splitting function with the highest splitting criteria is chosen. At each

branch, the new nodes are calculated in the same way. If a subjacent node does not improve the splitting criteria, then it becomes a leaf node. In the case of classification, the leaf node gets the class of the most frequent class at that node. In case of regression, it gets the average value of the samples at that node (Breiman 2017).

A decision tree is easy to understand, does not need much data preparation and it is a white box model. A big downside of decision trees is that they are prone to overfit the data (Svetnik et al. 2003). The bias of a decision tree is low, but the variance is high. A solution to the high variance are ensemble methods, like RF (Svetnik et al. 2003).

3.5.2 Random forest

Random forest (RF) is an ensemble of decision trees. RF was introduced by Breiman (2001). The prediction of the forest is a majority vote or the average of the predictions of the trees. Randomness at RF comes in two forms. Random subset of data and random subset of features to determine the splitting function (Liaw, Wiener, et al. 2002).

Bootstrapping is a sampling method where from a dataset a new dataset is constructed. The new dataset is constructed by drawing with replacement. Usually not all the samples end up in the new dataset and some samples can occur multiple times (Breiman 1996). Each tree in the RF is trained on a random subset of the dataset created by the bootstrap method (Liaw, Wiener, et al. 2002). To further reduce overfitting, not all the features are considered when splitting the data. At each node of each tree, a new subset of features is created. The size of this subset is usually the square root or the third of the number of features (Liaw, Wiener, et al. 2002).

3.6 Ensemble

Strong models can be constructed from weaker models, by combining their prediction. These methods are called ensembles. Ensembles work because of three reasons: statistical advantage, computational advantage, and representational advantage (Dietterich et al. 2002). The statistical advantage arises when the hypotheses space is too large for the available training data. The base learner can have several hypotheses with similar performance. By averaging these hypotheses risk can be reduced. The computational problem arises when a base learner gets stuck in local optima. Several learning algorithms can not guarantee to find the best solution, by combining these models, there is a higher chance to find the global optima. Last, a representation problem arises, when the unknown function

is not in the hypothesis space. By combining several of these models, the solution space can be increased (Dietterich et al. 2002). Ensembles work best if the base learners are diverse because vastly different models have a low correlation in their errors (Krauss, Do, and Huck 2017).

4 Methodology

In section 4.1, I describe the data used for the empirical evaluation, the hardware, and software used for computation and implementation. In section 4.2, you can read about the data transformation and feature generation. The methodology of the model training can be found in section 4.3. For the exact implementation go to my GitHub repository (`https://github.com/tillaczel/LSTM-GRU-RF-predicting-SP500`).

4.1 Data, hardware, software

To compare the models empirically, I chose the S&P 500 index. Fischer and Krauss (2018) and Krauss, Do, and Huck (2017) used the S&P 500 to compare the predicting power of machine learning models on financial time series data. The S&P 500 is a capitalization-weighted index constructed from 500 big companies listed on the US stock exchanges. The dataset consists of the closing prices from 2008 to 2017 at the minute by minute frequency.

The neural networks were trained on a GeForce GTX 1080 8 GB (*GEFORCE GTX 1080* 2018), while all the other models on an Intel Core i7-7820X 3.60GHz (*INTEL CORE i7-7820X PROCESSOR* 2018). For data manipulation I used the Pandas (McKinney 2011) and numpy (Oliphant 2006) libraries. The LSTM and GRU were implemented with tensorflow 2.0 (Abadi et al. 2015). The ARMA model with the pmdarima package (Smith et al. 2017), and the RF with scikit-learn (Pedregosa et al. 2011). For bayesian optimization I used the bayesian-optimization package (Nogueira 2019). The model confidence set was implemented with the help of the arch package (Sheppard et al. 2019).

4.2 Study periods and feature generation

First I resampled the data into five different frequencies: 1 minute, 5 minutes, 15 minutes, 1 hour and 1 day. All the following steps were done on each frequency. I split up the data into two year long study periods. Each study period consists of one year training,

half year validation and half year test set. Models that do not need a validation set for choosing hyperparameters were trained not only on the training but also on the validation set. The first study period starts at the beginning of the dataset, and the following study periods are shifted by half a year. That means that for the next study period the test set becomes the new validation set and the new test set does not overlap with the previous period. In total there are 16 study periods.

I constructed the logarithmic returns from the prices, and then except for the RF, I normalized the logarithmic returns. You can see the calculation of the logarithmic returns in equation 4.1, and of the normalization in equation 4.2. The mean and standard deviation for the normalization were calculated only on the training set.

$$r_t = \ln \frac{P_t}{P_{t-1}} \tag{4.1}$$

$$r_{\text{norm},t} = \frac{r_t - \text{mean}_{\text{train}}}{\text{std}_{\text{train}}} \tag{4.2}$$

I only used the logarithmic returns of past observations for the predictions, no additional features were constructed.

4.3 Training

In this section, I write about the specific methodology I used to define and train my models. At each frequency and study period, the hyperparameters were tuned using the validation set or the bayesian information criterion (Schwarz et al. 1978) and the models were trained on the training set.

4.3.1 Autoregressive moving average model

The ARMA model was implemented using the pmdarima library (Smith et al. 2017). For training I used the stepwise model selection (Hyndman, Khandakar, et al. 2007), with bayesian information criterion (Schwarz et al. 1978) and the Newton-Raphson solver (Gil, Segura, and Temme 2007) for the log-likelihood optimization. The MSE error function was used. The maximum order of autoregression and moving average was set to 5.

4.3.2 Recurrent neural networks

For the LSTM and GRU hyperparameter search I used bayesian hyperparameter optimization (Feurer and Hutter 2019) implemented by the bayesian-optimization library

(Nogueira 2019). The number of initialization points for the bayesian-optimization were 1 and the bayesian optimization run for 2 iterations. The boundaries for the hyperparameter search are in table 4.1. Some of the hyperparameters needed to be rounded. I set the batch size to only take on powers of two. The optimizers are stochastic gradient descent, RmsProp, and Adam. The network architecture is defined by two hyperparameters: first layer size and layer size decay. The first layer has the most neurons, and the coming layer sizes are multiplied by the layer size decay. The learning rate is chosen from an exponential scale. Early stopping was used. To have the same number of weight updates for each hyperparameter setting, the number of max epochs was proportional to the batch size.

hyperparameter	**minimum**	**maximum**
lookback	1	40
batch size	16	1024
optimizer	0	2
dropout	0	0.5
number of layers	1	4
first layer size	1	40
layer size decay	0.3	1
learning rate	$\frac{1}{e^0}$	$\frac{1}{e^{-15}}$

Table 4.1: Boundaries for the bayesian hyperparameter optimization

4.3.3 Random forest

Because all the other models are regressors, I chose to use the RF regression and not classification to predict the S&P 500 index. The RF was implemented with the help of the scikit-learn (Pedregosa et al. 2011) package. I set the maximum number of trees to 1024, the minimum number of samples in a node to 0.1% of the training dataset and as suggested by Geurts, Ernst, and Wehenkel (2006). I changed the number of features considered when determining the optimal split to 33% of the lookback. The lookback was optimized with bayesian optimization with boundaries of minimum 1 to maximum 40. Like by the LSTM and GRU the number of initialization points for the bayesian-optimization were 1 and the algorithm run for 2 iterations. Because for decision trees the inputs do not need to be normalized, the inputs for the RF were not normalized (Lewis

2000).

4.3.4 Ensemble

According to Krauss, Do, and Huck (2017), the equal weighted ensemble has a good performance on the S&P 500 compared to other more complicated ensembles. Because there is no significant advantage over using other ensembles I am using the equal weighted ensemble. The output of the equal weighted ensemble is the mean of the predictions of the base learners.

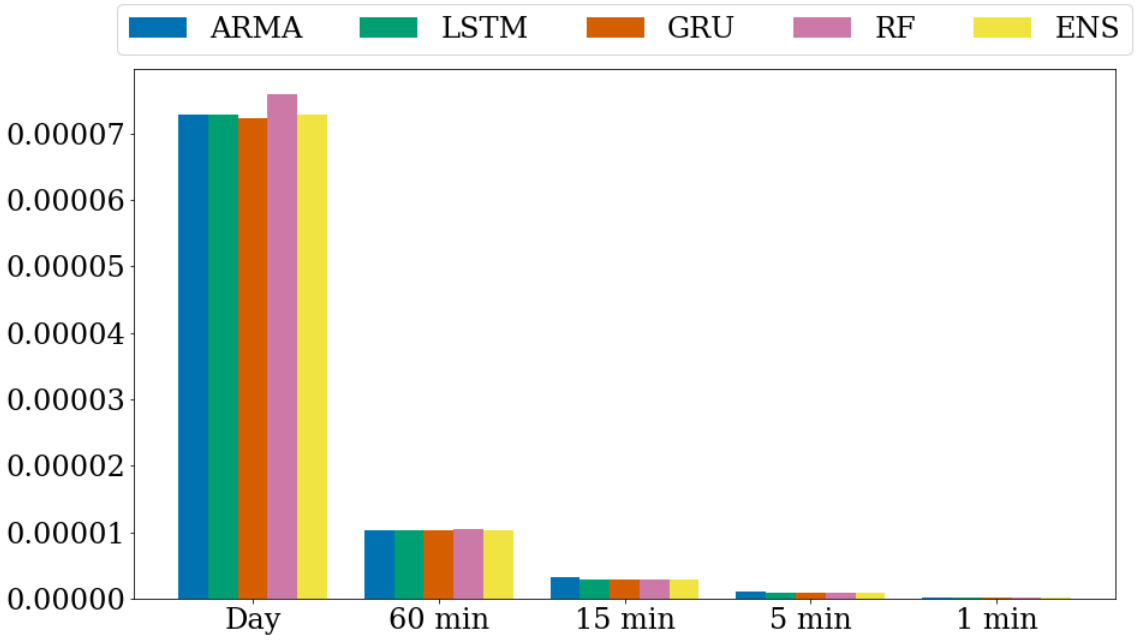
4.4 Forecast and trading

In the beginning, we have zero stocks. Each time step we can decide if we want to long, short or hold our current position on the S&P 500 index. There are two scenarios, one without transaction cost, and one with 5 base points of transaction cost. 5 base points of transaction cost is a conservative estimate. Like Krauss, Do, and Huck (2017) and Fischer and Krauss (2018) I chose 5 base points of transaction cost after Avellaneda and Lee (2010). When a trade happens if there is any transaction cost, the S&P return is adjusted with it. When to long, sort or hold our current position is decided for each model in the following way. For each frequency, the predictions on the training sets are concatenated together. If $c < R$ then the index is longed, if $R < -c$ then the index is shorted. If the predicted return falls between $-c$ and c then the previous position is kept, where c represents the transaction cost, and R the model's prediction for the next return calculated from the logarithmic return with the following equation 4.3.

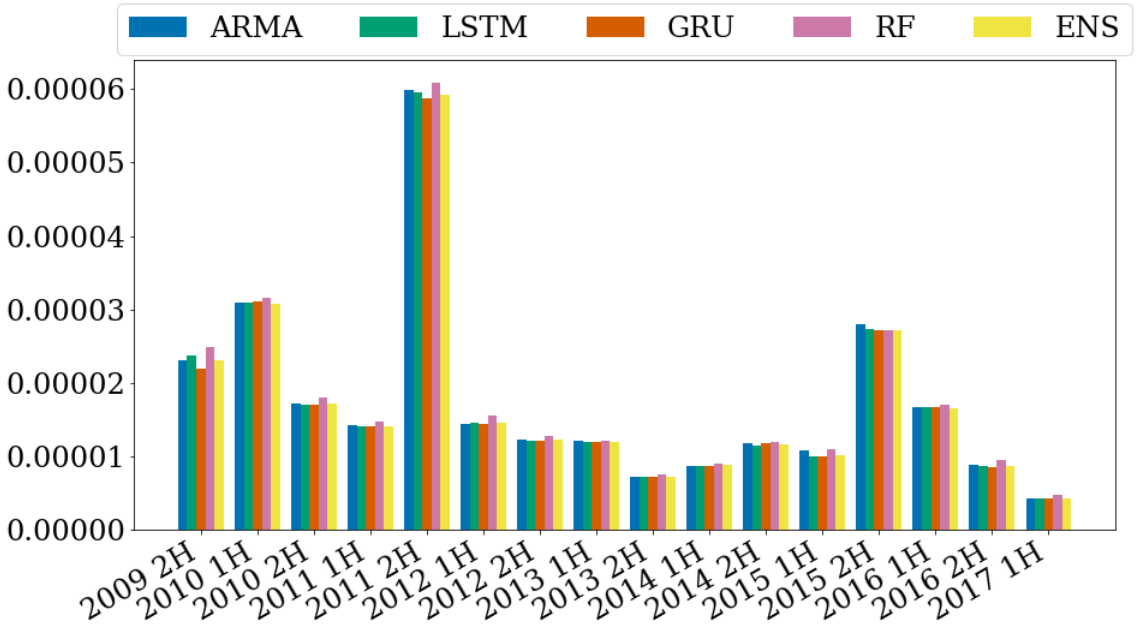
$$R = e^r - 1 \tag{4.3}$$

5 Results

I am presenting my results in two parts. First, I am analyzing the predictions, second I look at the performance of the trading strategies based on the model predictions. Because the minute by minute S&P 500 data is not public, in section 5.3 I apply and evaluate the same models on Bitcoin prices.



(a) Mean squared error over the different frequencies



(b) Mean squared error over the study periods

Figure 5.1: Mean squared error of ARMA, LSTM, GRU, RF and Ensemble model

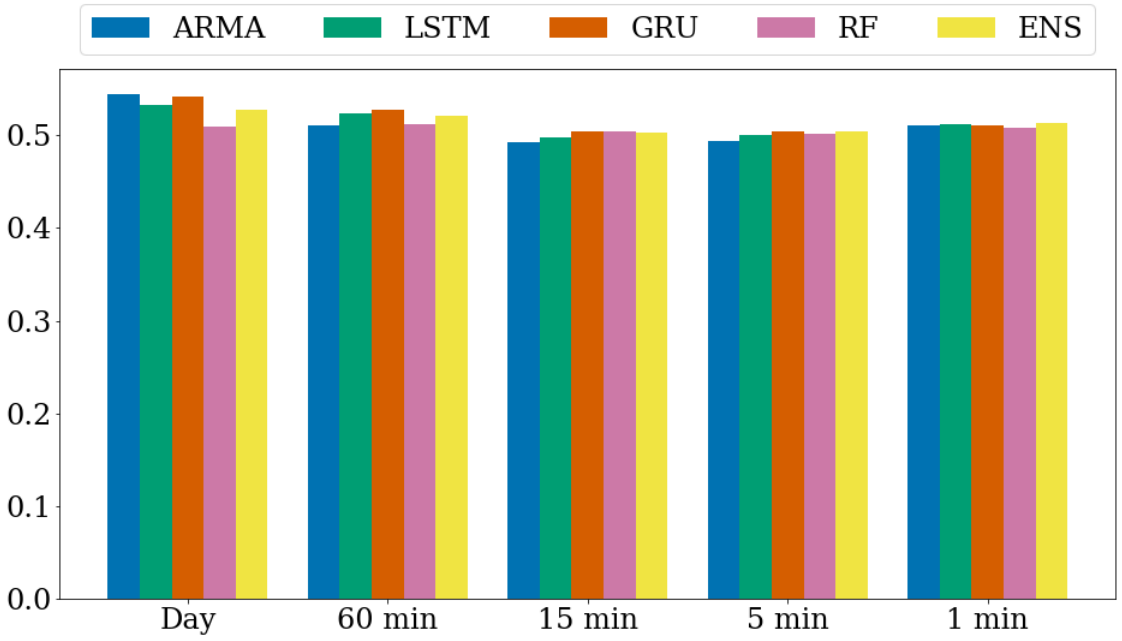
5.1 Predictions

MSE shows how inaccurate the predictions of a given model are. From figure 5.1 it is not obvious if there is a model that outperforms the other models. The average test set MSE over the different frequencies can be seen in figure 5.1a. As expected the MSE

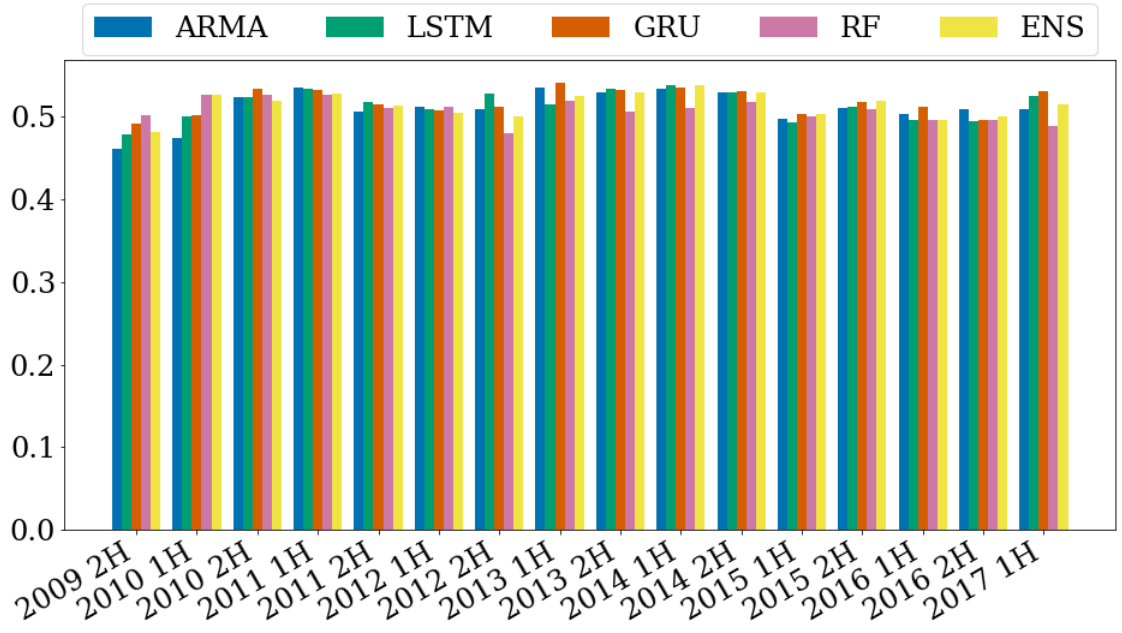
decreases, as the frequency increases. This is because there is a smaller price change in a minute than in a day, so the order of the MSE gets smaller. If we assume the S&P index follows a geometric brownian motion, then the standard deviation of the logarithmic return decreases with the squarer root of the frequency increase. The MSE decreases with the square of the standard deviation of the prediction error. This means, if the prediction quality stays the same over the different frequencies, it is true as a rule of thumb, that if the frequency gets increased by the factor of n, the MSE should decrease by the factor n. This can be observed in figure 5.1a. RF has slightly higher MSE at a daily frequency than the other models. As shown in figure 5.1b the MSE over the study periods exhibits a slightly negative trend. There are several study periods with outlining test set MSE, like the second half year of 2011 and 2015.

It can be seen in figure 5.2 that all the directional accuracy of all models is around 50%. There is no clear trend in directional accuracy with the change of the frequency. As shown in figure 5.1b directly after the 2008 financial crisis all the models have difficulties to predict the direction of the price movements, but in the long term, it stabilizes slightly over 50%.

The ENS model is more effective if the errors of the base models do not correlate. As shown in figure A.1 the model errors have an almost perfect positive correlation. Despite the high correlation the ENS model significantly outperforms all the other models at 1 minute frequency - see MCS next paragraph. This can be explained by the order of the predictions and the order of the return (table 5.1). The returns have a higher standard deviation than the predictions, so the prediction error is mostly defined by the returns. To see the differences between the predictions, figure 5.3 shows the correlation between the predictions and the S&P 500. The predictions of the base models are not strongly correlated, which means the ENS model can improve the prediction. Generally, the model predictions have medium positive correlation between each other and a small positive correlation with the S&P 500. As the frequency increases the correlation between the predictions and the S&P 500 increases. There is more information left in the past observations at a higher frequency, which means the markets are less effective. The ARMA model has almost no correlation with the LSTM, GRU, and RF at 60, 15 and 5 minutes frequency. Except for the 1 minute frequency, the ARMA model has the lowest correlation with the S&P 500. It can be concluded, that the relation between the past and the next logarithmic returns is more linear at the minute frequency. The high positive correlation between the LSTM and GRU is expected because they are similar models. The



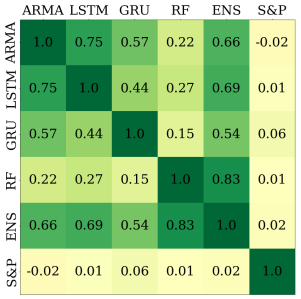
(a) Directional accuracy over the different frequencies



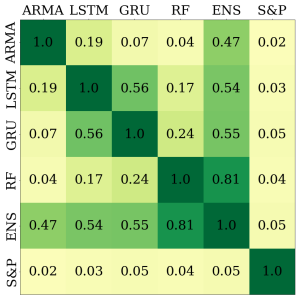
(b) Directional accuracy over the study periods

Figure 5.2: Directional accuracy of ARMA, LSTM, GRU, RF and Ensemble model

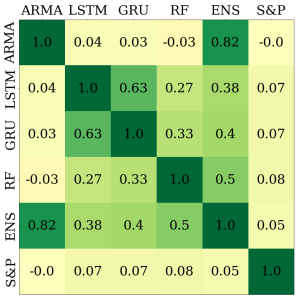
two models correlate higher than 0.5 except at a daily frequency where it is 0.44. This correlation is not 100% because there are many local minimums with similar minimum values. Because the ENS prediction is the average of the base models, models with a higher standard deviation have more effect on the ENS prediction. The models with higher standard deviation (table 5.1) have higher correlation with the ENS.



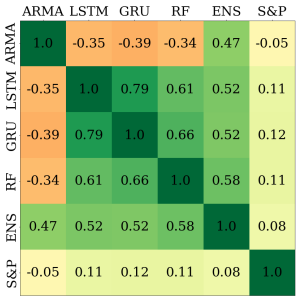
(a) Daily frequency



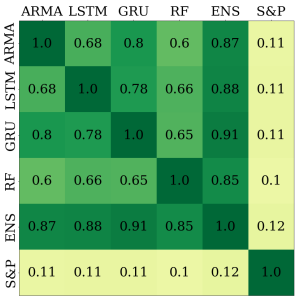
(b) 60 minutes frequency



(c) 15 minutes frequency



(d) 5 minutes frequency



(e) 1 minute frequency

Figure 5.3: Correlation matrix of model predictions and S&P at different frequencies

Frequency	**ARMA**	**LSTM**	**GRU**	**RF**	**ENS**	**S&P**
1 day	5.14e-04	7.30e-04	5.96e-04	1.89e-03	6.86e-04	8.51e-03
60 minutes	3.05e-04	1.70e-04	1.82e-04	5.60e-04	1.98e-04	3.21e-03
15 minutes	6.04e-04	1.08e-04	1.04e-04	3.23e-04	1.84e-04	1.68e-03
5 minutes	2.89e-04	1.01e-04	9.63e-05	1.69e-04	8.42e-05	9.68e-04
1 minute	4.45e-05	4.28e-05	4.15e-05	5.38e-05	4.00e-05	4.13e-04

Table 5.1: Standard deviation of model predictions and S&P 500 logarithmic returns.

The model confidence set (MCS) described by Hansen, Lunde, and Nason (2011) gives a set of models based on the model errors. This set of models contains the best model with a given probability. The output of the MCS is a set of p values that the given model is excluded from the set of best models. If the p value is 5%, the model is not in the set of best models on a 95% significance level. The higher a model's p value, the better the model. These p values are shown in table 5.2. The ARMA and RF are excluded from the best set of models at 95% significance at all frequency except the daily for ARMA and 15 minutes for RF. The LSTM is the best model at 15 minutes frequency, the GRU at daily

and 5 minutes frequency and the ENS at 60 and 1 minute frequency. At the 1 minute frequency, all models except the ENS have a p value below 5%, this means at a significance level of 95% the ENS model outperforms all base learners at a minute frequency. This finding is surprising if we only look at the mean MSE of the models. As shown in table A.1 the MSE of the ENS at 1 minute frequency is just slightly lower than of the base models.

Frequency	**ARMA**	**LSTM**	**GRU**	**RF**	**ENS**
1 day	0.0650	0.1480	1.0000	0.0000	0.1480
60 minutes	0.0420	0.3720	0.6870	0.0000	1.0000
15 minutes	0.0020	1.0000	0.7220	0.1590	0.0910
5 minutes	0.0000	0.4590	1.0000	0.0060	0.0000
1 minute	0.0000	0.0020	0.0070	0.0000	1.0000

Table 5.2: Model confidence set

5.2 Performane of the trading strategy

The sum logarithmic return for the S&P 500 index and all the strategies based on the model predictions can be seen without transaction cost in table 5.3 and with 5 base points of transaction cost in table 5.4. The differences between the logarithmic returns of the S&P index at different frequencies are due to the slightly shifted test sets, because of the different intra-day start and end time. First, let us look at the model performances without transaction cost. At daily frequency, all the models underperform the S&P 500 index. The returns of the trading strategy based on the AMRA model are lower than the S&P 500 at all except the minute by minute frequency. At 5 minutes frequency the ARMA model is worse than a random model. The GRU outperforms the LSTM at all frequencies. GRU, RF, and ENS achieve better returns than the S&P 500 at all except the daily frequency. At 1 minute frequency, all models achieve a logarithmic return of more than 10 times of the S&P 500. The model returns would be hard to predict only from the directional accuracies at different frequencies. As shown in table 5.4, with the introduction of transaction cost the advantage of the trading strategies based on the model predictions diminishes. This is especially true at the 1 minute frequency. The logarithmic returns at the minute by minute frequency are just slightly higher, or around the S&P 500. At 1 day, 60 minutes, 15 minutes and 5 minutes frequency all models underperform

the S&P 500, with except for the GRU at 5 minutes frequency. The returns of the ARMA model at 15 minutes and 5 minutes frequency and the RF at 1 day, 60 minutes and 15 minutes frequency are negative.

Frequency	**ARMA**	**LSTM**	**GRU**	**RF**	**ENS**	**S&P**
1 day	0.67	0.51	0.82	0.29	0.37	0.97
60 minutes	0.69	0.94	1.57	1.08	1.20	0.98
15 minutes	0.14	0.94	1.13	1.44	1.54	0.97
5 minutes	-1.24	2.73	3.04	2.04	2.28	0.98
1 minute	14.88	13.60	13.49	11.14	15.65	0.98

Table 5.3: Sum logarithmic return with no transaction cost

Frequency	**ARMA**	**LSTM**	**GRU**	**RF**	**ENS**	**S&P**
1 day	0.66	0.41	0.86	-0.57	0.79	0.97
60 minutes	0.35	0.28	0.52	-0.72	0.78	0.98
15 minutes	-0.22	0.55	0.38	-0.29	0.66	0.97
5 minutes	-3.22	0.70	1.14	0.70	0.86	0.98
1 minute	1.12	1.48	0.97	1.03	1.19	0.98

Table 5.4: Sum logarithmic return with 5 base points of transaction cost

In figure 5.4 the cumulative logarithmic returns and the cumulative number of trades without transaction cost can be seen. It must be noted, that except for the first trade all trades account for two trades. The first trade is switching from no position to long or short (selling or buying one). Every trade after the first is either a switch from long to short or reversed, which requires two trades. At daily and 60 minutes frequency in the first two study periods, all the models achieve a negative logarithmic return. At 15 and 5 minutes frequency, the ARMA logarithmic return is much lower than the other model's returns. This was expected from the correlation matrix. At 1 minute frequency, the cumulative logarithmic return increases monotonically until the last study period for all the models except RF. In 2017 no model was able to outperform the S&P 500 index. From 2014 January the RF did not make any trades for 1.5 years and after that only a few in the coming half year. RF changed its position the most for all except the 1 minute frequency.

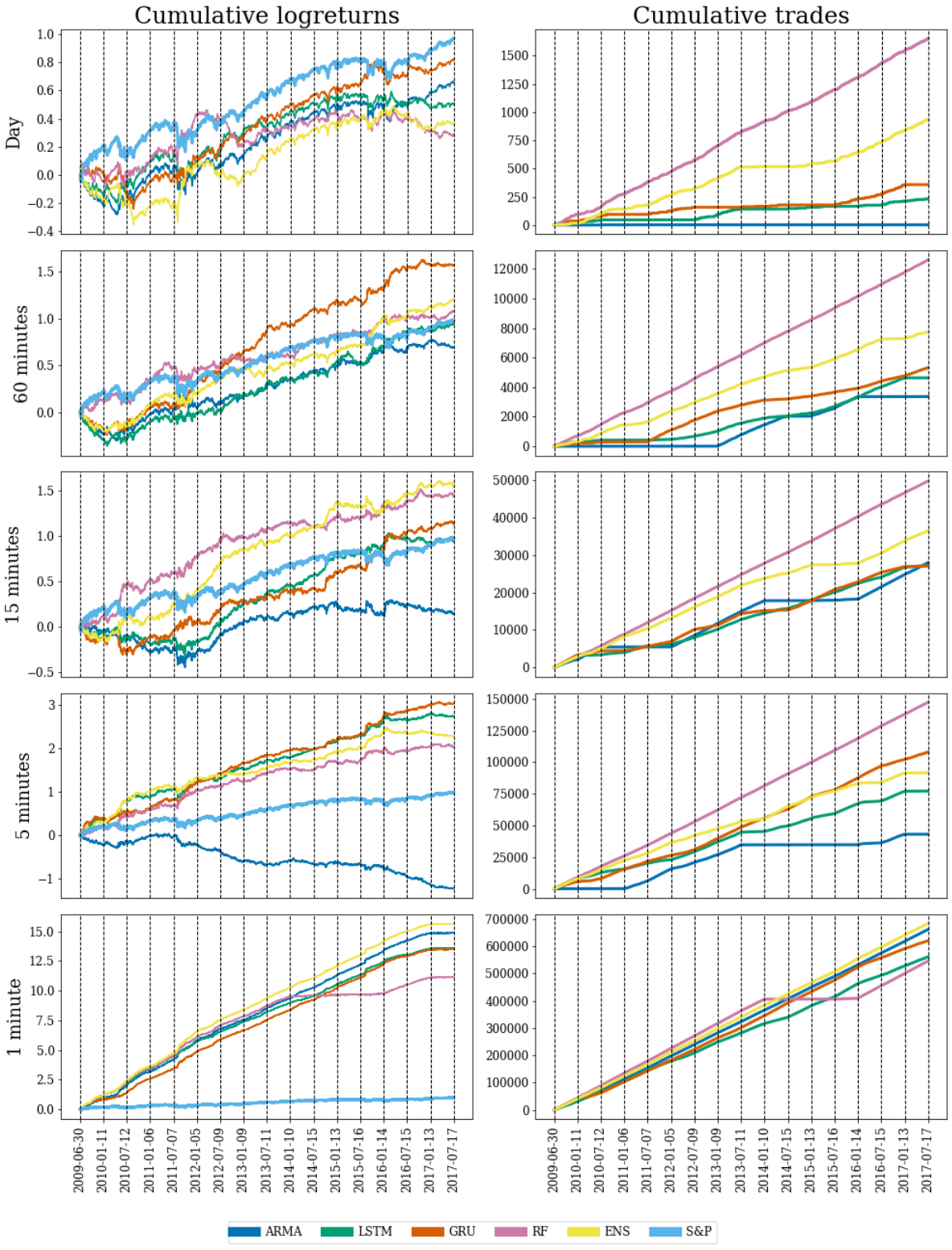


Figure 5.4: Cumulative logarithmic return and cumulative sum of trades of ARMA, LSTM, GRU, RF, Ensemble model and S&P 500 index at different frequencies with no transaction cost

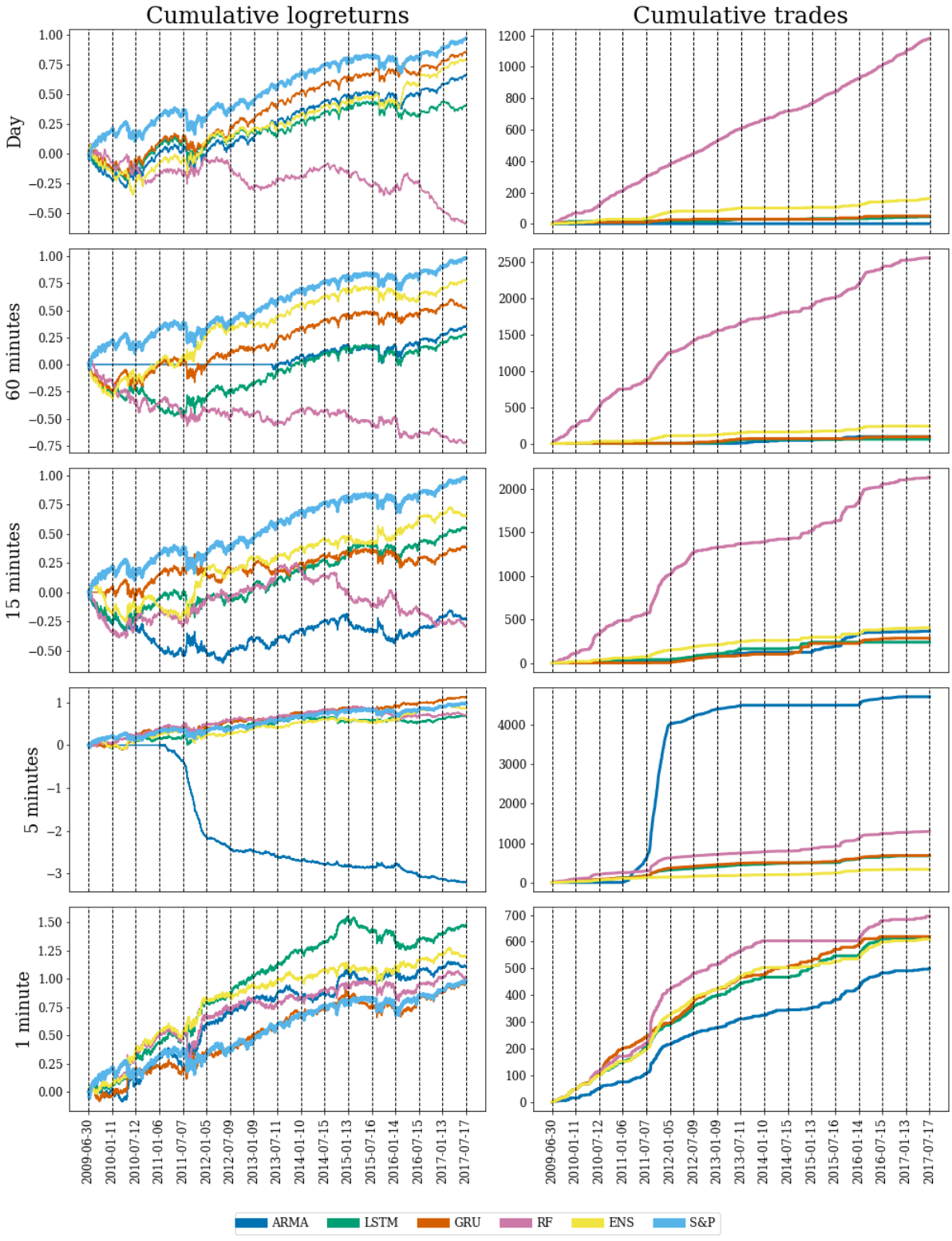


Figure 5.5: Cumulative logarithmic return and cumulative sum of trades of ARMA, LSTM, GRU, RF, Ensemble model and S&P 500 index at different frequencies with 5 base points of transaction cost

Figure 5.5 shows the cumulative logarithmic returns and the cumulative number of trades with 5 base points of transaction cost. It can be seen that RF is overconfident with its predictions at daily to 15 minutes frequency. The trading strategy based on RF makes too may trades and the profit is lost to the transaction cost. At 60 minutes frequency, the ARMA model does not make any trades until the second half year of 2013. At 5 minutes frequency in 2011 the ARMA model makes 4000 trades. The transaction cost of these trades plummets its logarithmic returns below -2, which corresponds to a loss of almost 90% of the original capital. After 2015 the slope of the upward trend in the cumulative logarithmic return diminishes.

Frequency	**ARMA**	**LSTM**	**GRU**	**RF**	**ENS**	**S&P**
1 day	0.69	0.54	0.83	0.34	0.41	0.97
60 minutes	0.71	0.94	1.52	1.08	1.19	0.98
15 minutes	0.20	0.94	1.11	1.39	1.48	0.96
5 minutes	-1.08	2.60	2.88	1.95	2.17	0.98
1 minute	14.50	13.26	13.15	10.87	15.25	1.02

Table 5.5: Annualized sharpe ratio with no transaction cost

Frequency	**ARMA**	**LSTM**	**GRU**	**RF**	**ENS**	**S&P**
1 day	0.69	0.45	0.87	-0.46	0.81	0.97
60 minutes	0.59	0.33	0.55	-0.60	0.80	0.98
15 minutes	-0.14	0.57	0.43	-0.20	0.68	0.96
5 minutes	-3.40	0.72	1.12	0.72	0.86	0.98
1 minute	1.15	1.50	1.00	1.06	1.22	1.02

Table 5.6: Annualized sharpe ratio with 5 base points of transaction cost

The sharpe ratio (Sharpe 1966) measures excess return per standard deviation (Krauss, Do, and Huck 2017). The Sharpe ratios of the models and the S&P 500 without and with transaction cost can be seen in table 5.5 and 5.6. The sharpe ratio was calculated with 0% risk-free rate. Because the standard deviation of the S&P 500 was relatively lower at the minute by minute frequency, the excess return over risk got relatively to the other frequencies higher. The sharpe ratio of the models with no transaction cost is like the logarithmic return, more than 10 times higher than the sharpe ratio of the S&P 500.

Before transaction cost at 5 minutes and 1 minute frequencies except for the ARMA model, all models have a sharpe ratio of almost two or more than two compared to the 1.02 of the S&P 500. The GRU, RF, and ENS have a higher sharpe ratio than the S&P at all except the daily frequency. After transaction cost (see table 5.6) almost all the models outperform the S&P 500 at 1 minute frequency.

The sortino ratio like the sharpe ratio measures excess return over unit of risk. In contrast to the sharpe ratio, the risk metric is the downside deviation. The advantage of the sortino ratio is that it does not penalize for positive deviations (Krauss, Do, and Huck 2017). Like the sharpe ratio, the sortino ratio was also calculated with 0% risk-free rate. If the models traded too much, because of the transaction cost, the sortino ratio of the models should relatively be worse than the sortino ratio of the S&P 500 compared to their sharpe ratio. The trading strategies based on the model predictions did not get relatively worse compared to the sharpe ratio.

Frequency	**ARMA**	**LSTM**	**GRU**	**RF**	**ENS**	**S&P**
1 day	0.95	0.74	1.19	0.47	0.57	1.30
60 minutes	1.01	1.31	2.16	1.51	1.68	1.36
15 minutes	0.28	1.33	1.56	2.05	2.16	1.33
5 minutes	-1.48	3.74	4.13	2.81	3.13	1.35
1 minute	20.01	18.11	17.98	14.68	21.01	1.36

Table 5.7: Annualized sortino ratio with no transaction cost

Frequency	**ARMA**	**LSTM**	**GRU**	**RF**	**ENS**	**S&P**
1 day	0.95	0.62	1.22	-0.62	1.15	1.30
60 minutes	0.84	0.46	0.77	-0.84	1.11	1.36
15 minutes	-0.20	0.81	0.60	-0.29	0.97	1.33
5 minutes	-4.60	1.01	1.59	1.01	1.22	1.35
1 minute	1.54	2.01	1.34	1.41	1.63	1.36

Table 5.8: Annualized sortino ratio with 5 base points of transaction cost

In summary, there it is not clear which model is the best in all scenarios. ENS model has significantly better predictions than all the other models at 1 minute frequency. At all of the other frequencies, no model significantly outperforms all the other models. At

95% significance level the ARMA model is rejected from the set of best models at all frequency except the daily frequency and the RF at all except 15 minutes frequency. The MSE of the RF is worse than all the other model MSE. The average directional accuracy is above 51% for all the models. At 1 minute frequency, all the models have a strong positive correlation with each other. This strong correlation is due to the ability of the models to approximate the underlying function well. At 1 minute frequency, there is enough information in the past observations to predict the next return. This can also be observed in the strategy returns. At 1 minute frequency, all the models outperform the S&P 500 index in the means of returns, sharpe ratio, and also in the sortino ratio. The RF took roughly three times as much and the LSTM and GRU eight times as much time to train than the ARMA model.

5.3 Bitcoin

Because Bitcoin is 100% traded electronically, it has high-quality data. I applied the same methodology as used for the S&P 500 with small adjustments to the past two years of Bitcoin prices. I downloaded the data from Kaggle (Zielak 2019). Because Bitcoin is traded 24/7 at all frequencies except daily there is roughly the same amount of data in two years of Bitcoin prices as in eight years of S&P prices. Because I did not change the frequencies, there is a big gap between the daily and hourly frequency. To have about the same amount of data in one study period, I decreased the size of a study period from two years to four months. The figures and tables of the Bitcoin results are provided in appendix B.

In the rest of this section, I am listing some of the differences between the results on the S&P 500 and Bitcoin data. The LSTM achieves the lowest MSE at all frequencies. The correlation between the predictions is the highest at daily frequency. The standard deviation of the Bitcoin logarithmic returns and so of the model predictions is higher. At 95% significance level at 5 minutes frequency, only the LSTM is in the set of best models. At 1 minute frequency without transaction cost, except for the ARMA model, all models achieve a sum logarithmic return of higher than 50. With transaction cost, the ARMA model has negative return at all except for the daily frequency. At 1 minute frequency with 5 base points of transaction cost, the LSTM has 9.09, the GRU 7.90, the RF 5.17, and the ENS 4.18 sum logarithmic return. In the first few study periods, most of the models vastly outperform the Bitcoin returns. With transaction cost, the LSTM

model has the highest sharpe and sortino ratio at all frequencies. In summary, the LSTM outperformed all the other models in almost all of the scenarios. The ARMA model performed the worst. The GRU had slightly better results than the RF and ENS. This implies there is much higher nonlinearity in the Bitcoin returns than in the S&P returns.

6 Conclusion and future work

At one minute frequency, the market is not efficient, there is information in the past observations. At lower frequencies, these models are not capable to predict the returns. The ENS model outperforms the other models in predictive capabilities at 1 minute frequency. The LSTM and GRU have slightly better MSE than the ARMA and GRU. This suggests, that the underlying function has some nonlinearity. Regarding PnL of the trading strategy, there is no model, which outperforms the other models at any of the frequencies. The markets get slightly more efficient after 2015.

LSTM, GRU, and RF are more computationally expensive than the ARMA model. With further hyperparameter search, LSTM and GRU might achieve better results. The LSTM and GRU model could be upgraded with more advanced architectures like attention (Vaswani et al. 2017). The bayesian hyperparameter search could be improved by changing the correlation function to allow for the rounding of the discrete parameters. Only the past returns were used to predict future returns. Other features could be introduced by switching the ARMA model to an ARMA model with exogenous inputs (ARMAX) and by expanding the feature space for the other models. New features could be the time of day, time of year, volume, US bond yield curve, oil price, etc. Both for features and prediction I used middle price returns. In a more realistic scenario, the trading would not happen on the middle price, but there would be a bid-ask spread. After predicting the bid-ask price, a trading strategy would need to be formed. This could be evaded by using an end-to-end reinforcement learning model. High-frequency trading uses tick data and not minute by minute data. Provided tick data it could be investigated if the model performances further improve at higher frequencies. It should also be noted, that at high-frequency trading the speed of the model predictions affects the profitability of the models.

Even though some of the models achieved higher returns than the market it is unlikely that the models could consistently hold these returns. The transaction cost was set conservatively and the model performances got worse in the last study periods. A team

with more computing power could improve a model, which outperforms the market today. With the introduction of a better model the market efficiency increases and in the long run the model becomes useless.

References

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org, Accessed: 2019-10-15. URL: `http://tensorflow.org/`.

Aczél, Till (2019). *Predicting the S&P 500 with long short-term memory network, gated recurrent unit and random forest*. Accessed: 2019-10-15. URL: `https://github.com/tillaczel/LSTM-GRU-RF-predicting-SP500`.

Akaike, Hirotogu (1998). "Information theory and an extension of the maximum likelihood principle". In: *Selected papers of hirotugu akaike*. Springer, pp. 199–213.

Aldridge, Irene and Steven Krawciw (2017). *Real-time risk: What investors should know about FinTech, high-frequency trading, and flash crashes*. John Wiley & Sons.

Alonso, Miquel N, Gilberto Batres-Estrada, and Aymeric Moulin (2018). "Deep Learning in Finance: Prediction of Stock Returns with Long Short-Term Memory Networks". In: *Big Data and Machine Learning in Quantitative Investment*, pp. 251–277.

Altrichter, Márta et al. (2006). *Neurális hálózatok*. Panem, Budapest.

Avellaneda, Marco and Jeong-Hyun Lee (2010). "Statistical arbitrage in the US equities market". In: *Quantitative Finance* 10.7, pp. 761–782.

Bengio, Yoshua (1991). "Artificial neural networks and their application to sequence recognition". PhD thesis. McGill University.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Bergstra, James and Yoshua Bengio (2012). "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb, pp. 281–305.

Borovkova, Svetlana and Ioannis Tsiamas (2018). "An ensemble of LSTM neural networks for high-frequency stock market classification". In: *Journal of Forecasting*.

Box, George EP et al. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Breiman, Leo (1996). "Bagging predictors". In: *Machine learning* 24.2, pp. 123–140.

Breiman, Leo (2001). "Random forests". In: *Machine learning* 45.1, pp. 5–32.

Breiman, Leo (2017). *Classification and regression trees*. Routledge.

Brockwell, Peter J, Richard A Davis, and Stephen E Fienberg (1991). *Time Series: Theory and Methods*. Springer Science & Business Media.

Brooks, Chris (May 2008). *Brooks' Introductory Econometrics for Finance - Solutions to end of chapter questions*. Accessed: 2018-12-15. URL: https://www.cambridge.org/features/economics/brooks/Solutions.html.

Cho, Kyunghyun et al. (2014). "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259*.

Chung, Junyoung et al. (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555*.

Dietterich, Thomas G et al. (2002). "Ensemble learning". In: *The handbook of brain theory and neural networks* 2, pp. 110–125.

Elman, Jeffrey L (1990). "Finding structure in time". In: *Cognitive science* 14.2, pp. 179–211.

Feurer, Matthias and Frank Hutter (2019). "Hyperparameter optimization". In: *Automated Machine Learning*. Springer, pp. 3–33.

Fischer, Thomas and Christopher Krauss (2018). "Deep learning with long short-term memory networks for financial market predictions". In: *European Journal of Operational Research* 270.2, pp. 654–669.

GEFORCE GTX 1080 (2018). Accessed: 2019-09-28. URL: https://www.nvidia.com/en-gb/geforce/products/10series/geforce-gtx-1080/.

Geurts, Pierre, Damien Ernst, and Louis Wehenkel (2006). "Extremely randomized trees". In: *Machine learning* 63.1, pp. 3–42.

Gil, Amparo, Javier Segura, and Nico Temme (Jan. 2007). *Numerical Methods for Special Functions*. ISBN: 978-0-89871-634-4.

Goodfellow, Ian et al. (2016). *Deep learning*. Vol. 1. MIT press Cambridge.

Hansen, Peter R, Asger Lunde, and James M Nason (2011). "The model confidence set". In: *Econometrica* 79.2, pp. 453–497.

Haykin, Simon S et al. (2009). *Neural networks and learning machines*. Vol. 3. Pearson Upper Saddle River.

Hochreiter, Sepp (1991). "Untersuchungen zu dynamischen neuronalen Netzen". MA thesis. Technische Universität München.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5, pp. 359–366.

Hughes, Dana and Nikolaus Correll (June 2016). "Distributed Machine Learning in Materials that Couple Sensing, Actuation, Computation and Communication". In: *arXiv preprint arXiv:1606.03508*.

Hyndman, Rob J and George Athanasopoulos (2018). *Forecasting: principles and practice*. OTexts.

Hyndman, Rob J, Yeasmin Khandakar, et al. (2007). *Automatic time series for forecasting: the forecast package for R*. 6/07. Monash University, Department of Econometrics and Business Statistics.

INTEL CORE i7-7820X PROCESSOR (2018). Accessed: 2019-09-28. URL: `https://www.intel.co.uk/content/www/uk/en/products/processors/core/x-series/i7-7820x.html`.

Jin, Wen et al. (2000). "The improvements of BP neural network learning algorithm". In: *Signal processing proceedings, 2000. WCCC-ICSP 2000. 5th international conference on*. Vol. 3. IEEE, pp. 1647–1649.

Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever (2015). "An empirical exploration of recurrent network architectures". In: *International Conference on Machine Learning*, pp. 2342–2350.

Krauss, Christopher, Xuan Anh Do, and Nicolas Huck (2017). "Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500". In: *European Journal of Operational Research* 259.2, pp. 689–702.

Lewis, Roger J (2000). "An introduction to classification and regression tree (CART) analysis". In: *Annual meeting of the society for academic emergency medicine in San Francisco, California*. Vol. 14.

Liaw, Andy, Matthew Wiener, et al. (2002). "Classification and regression by randomForest". In: *R news* 2.3, pp. 18–22.

Lin, Henry W, Max Tegmark, and David Rolnick (2017). "Why does deep and cheap learning work so well?" In: *Journal of Statistical Physics* 168.6, pp. 1223–1247.

Lipton, Zachary C, John Berkowitz, and Charles Elkan (2015). "A critical review of recurrent neural networks for sequence learning". In: *arXiv preprint arXiv:1506.00019*.

Makridakis, Spyros and Michele Hibon (1997). "ARMA models and the Box–Jenkins methodology". In: *Journal of Forecasting* 16.3, pp. 147–163.

Malkiel, Burton G and Eugene F Fama (1970). "Efficient capital markets: A review of theory and empirical work". In: *The journal of Finance* 25.2, pp. 383–417.

McKinney, Wes (2011). "pandas: a foundational Python library for data analysis and statistics". In: *Python for High Performance and Scientific Computing* 14.

Neusser, Klaus et al. (2016). *Time series econometrics*. Springer.

Nogueira, Fernando (2019). *Bayesian Optimization*. Accessed: 2019-09-12. URL: https://github.com/fmfn/BayesianOptimization.

Olah, Christopher (Aug. 2015). *Understanding LSTM Networks - Colah's blog*. Accessed: 2018-12-16. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Oliphant, Travis E (2006). *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, p. 533.

Saunders, Daniel (July 2017). *The Bias-Variance Tradeoff*. Accessed: 2018-12-16. URL: https://djsaunde.wordpress.com/2017/07/17/the-bias-variance-tradeoff/.

Schwarz, Gideon et al. (1978). "Estimating the dimension of a model". In: *The annals of statistics* 6.2, pp. 461–464.

Sharpe, William F (1966). "Mutual fund performance". In: *The Journal of business* 39.1, pp. 119–138.

Shen, Guizhu et al. (2018). "Deep learning with gated recurrent unit networks for financial sequence predictions". In: *Procedia computer science* 131, pp. 895–903.

Sheppard, Kevin et al. (Aug. 2019). *bashtage/arch: Release 4.9.1*. Version 4.9.1. Accessed: 2019-10-15. DOI: 10.5281/zenodo.3382482. URL: https://doi.org/10.5281/zenodo.3382482.

Siami-Namini, Sima and Akbar Siami Namin (2018). "Forecasting Economics and Financial Time Series: ARIMA vs. LSTM". In: *arXiv preprint arXiv:1803.06386*.

Smith, Taylor G. et al. (2017). *pmdarima: ARIMA estimators for Python*. Accessed: 2019-09-12. URL: http://www.alkaline-ml.com/pmdarima.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Sutskever, Ilya (2013). "Training recurrent neural networks". PhD thesis. University of Toronto Toronto, Ontario, Canada.

Svetnik, Vladimir et al. (2003). "Random forest: a classification and regression tool for compound classification and QSAR modeling". In: *Journal of chemical information and computer sciences* 43.6, pp. 1947–1958.

Tashman, Leonard J (2000). "Out-of-sample tests of forecasting accuracy: an analysis and review". In: *International journal of forecasting* 16.4, pp. 437–450.

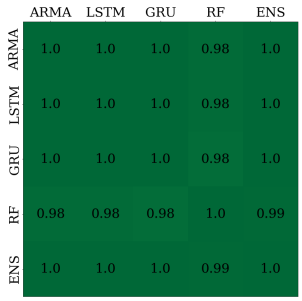
Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008.

Williams, Ronald J and Jing Peng (1990). "An efficient gradient-based algorithm for online training of recurrent network trajectories". In: *Neural computation* 2.4, pp. 490–501.

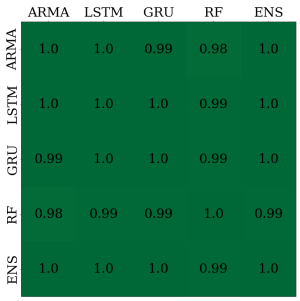
Zhang, G Peter, B Eddy Patuwo, and Michael Y Hu (2001). "A simulation study of artificial neural networks for nonlinear time-series forecasting". In: *Computers & Operations Research* 28.4, pp. 381–396.

Zielak (2019). *Bitcoin Historical Data*. Accessed: 2019-10-15. URL: `https://www.kaggle.com/mczielinski/bitcoin-historical-data`.

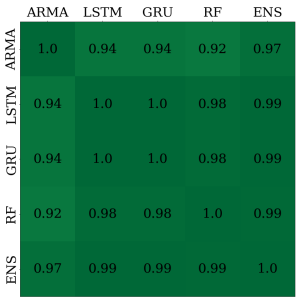
A Appendix



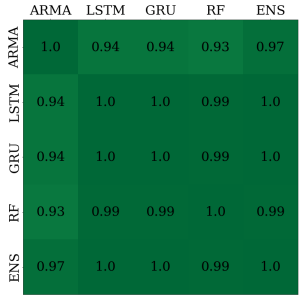
(a) Daily frequency



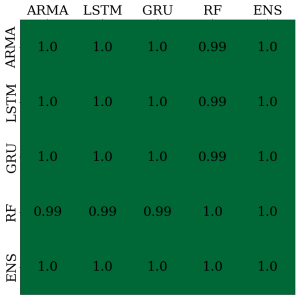
(b) 60 minutes frequency



(c) 15 minutes frequency



(d) 5 minutes frequency



(e) 1 minute frequency

Figure A.1: Correlation matrix of model errors at different frequencies

Frequency	ARMA	LSTM	GRU	RF	ENS	Mean
1 day	7.29e-05	7.29e-05	7.22e-05	7.59e-05	7.28e-05	7.33e-05
60 minutes	1.04e-05	1.03e-05	1.03e-05	1.05e-05	1.03e-05	1.04e-05
15 minutes	3.25e-06	2.82e-06	2.82e-06	2.86e-06	2.84e-06	2.92e-06
5 minutes	1.05e-06	9.25e-07	9.24e-07	9.28e-07	9.30e-07	9.51e-07
1 minute	1.69e-07	1.69e-07	1.69e-07	1.69e-07	1.68e-07	1.69e-07
Mean	1.75e-05	1.74e-05	1.73e-05	1.81e-05	1.74e-05	1.75e-05

Table A.1: MSE over the frequencies

Study period	ARMA	LSTM	GRU	RF	ENS	Mean
2009 H2	2.30e-05	2.38e-05	2.20e-05	2.49e-05	2.32e-05	2.34e-05
2010 H1	3.09e-05	3.09e-05	3.10e-05	3.16e-05	3.08e-05	3.11e-05
2010 H2	1.71e-05	1.71e-05	1.71e-05	1.80e-05	1.72e-05	1.73e-05
2011 H1	1.42e-05	1.42e-05	1.42e-05	1.48e-05	1.42e-05	1.43e-05
2011 H2	5.98e-05	5.95e-05	5.88e-05	6.09e-05	5.92e-05	5.96e-05
2012 H1	1.45e-05	1.45e-05	1.44e-05	1.56e-05	1.47e-05	1.48e-05
2012 H2	1.23e-05	1.21e-05	1.22e-05	1.28e-05	1.23e-05	1.23e-05
2013 H1	1.22e-05	1.20e-05	1.21e-05	1.21e-05	1.20e-05	1.21e-05
2013 H2	7.22e-06	7.26e-06	7.30e-06	7.66e-06	7.30e-06	7.35e-06
2014 H1	8.81e-06	8.76e-06	8.81e-06	9.14e-06	8.82e-06	8.87e-06
2014 H2	1.19e-05	1.15e-05	1.18e-05	1.20e-05	1.16e-05	1.17e-05
2015 H1	1.08e-05	1.01e-05	1.01e-05	1.11e-05	1.03e-05	1.05e-05
2015 H2	2.80e-05	2.73e-05	2.71e-05	2.72e-05	2.71e-05	2.73e-05
2016 H1	1.68e-05	1.68e-05	1.68e-05	1.70e-05	1.66e-05	1.68e-05
2016 H2	8.94e-06	8.67e-06	8.62e-06	9.51e-06	8.78e-06	8.91e-06
2017 H1	4.35e-06	4.33e-06	4.28e-06	4.79e-06	4.34e-06	4.42e-06
Mean	1.75e-05	1.74e-05	1.73e-05	1.81e-05	1.74e-05	1.75e-05

Table A.2: MSE over the study periods

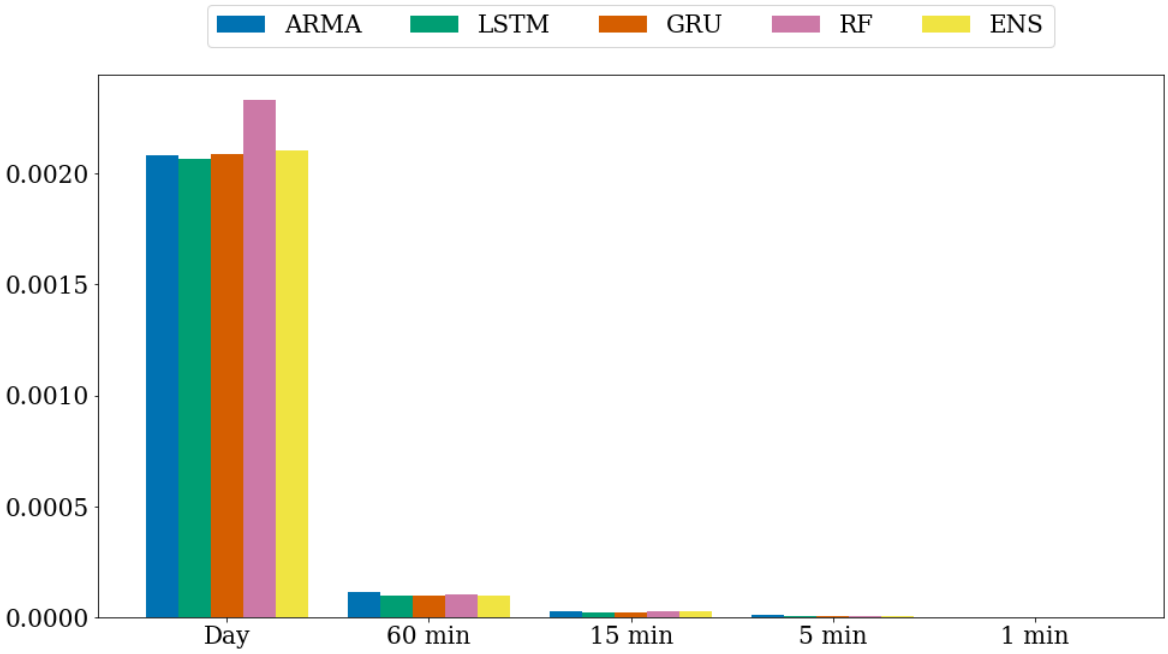
Frequency	ARMA	LSTM	GRU	RF	ENS	Mean
1 day	0.54	0.53	0.54	0.51	0.53	0.53
60 minutes	0.51	0.52	0.53	0.51	0.52	0.52
15 minutes	0.49	0.50	0.50	0.50	0.50	0.50
5 minutes	0.49	0.50	0.50	0.50	0.50	0.50
1 minute	0.51	0.51	0.51	0.51	0.51	0.51
Mean	0.51	0.51	0.52	0.51	0.51	0.51

Table A.3: Directional accuracy over the frequencies

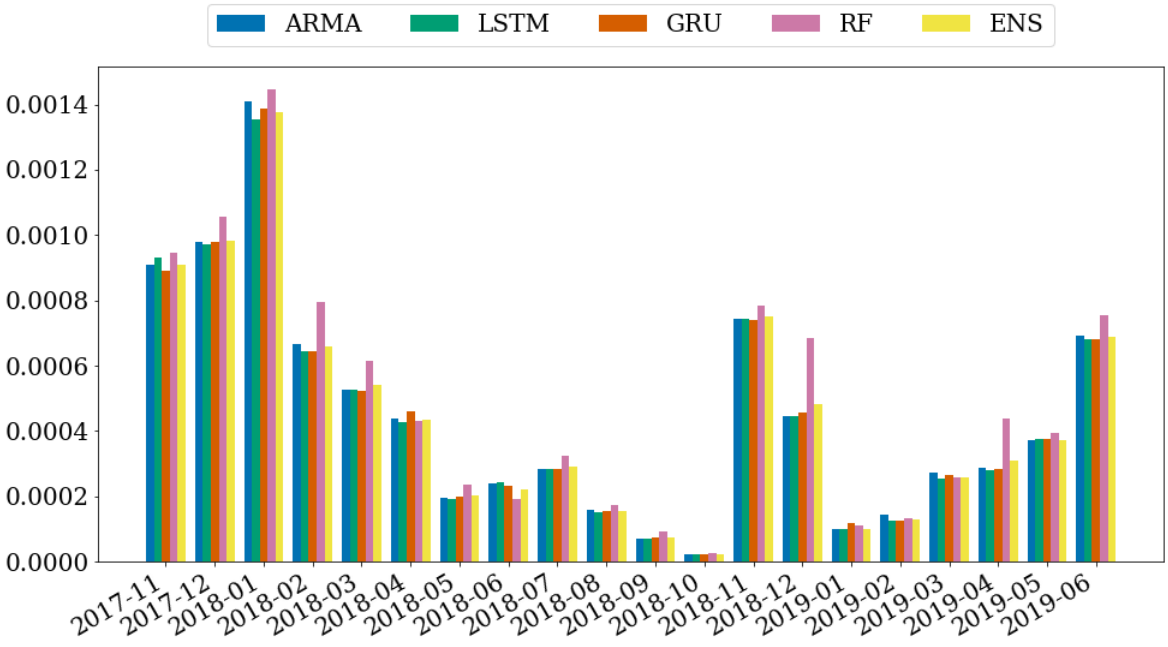
Study period	ARMA	LSTM	GRU	RF	ENS	Mean
2009 H2	0.46	0.48	0.49	0.50	0.48	0.48
2010 H1	0.47	0.50	0.50	0.53	0.53	0.51
2010 H2	0.52	0.52	0.53	0.53	0.52	0.52
2011 H1	0.53	0.53	0.53	0.53	0.53	0.53
2011 H2	0.51	0.52	0.51	0.51	0.51	0.51
2012 H1	0.51	0.51	0.51	0.51	0.50	0.51
2012 H2	0.51	0.53	0.51	0.48	0.50	0.51
2013 H1	0.53	0.51	0.54	0.52	0.52	0.53
2013 H2	0.53	0.53	0.53	0.51	0.53	0.53
2014 H1	0.53	0.54	0.53	0.51	0.54	0.53
2014 H2	0.53	0.53	0.53	0.52	0.53	0.53
2015 H1	0.50	0.49	0.50	0.50	0.50	0.50
2015 H2	0.51	0.51	0.52	0.51	0.52	0.51
2016 H1	0.50	0.50	0.51	0.50	0.50	0.50
2016 H2	0.51	0.49	0.49	0.50	0.50	0.50
2017 H1	0.51	0.52	0.53	0.49	0.51	0.51
Mean	0.51	0.51	0.52	0.51	0.51	0.51

Table A.4: Directional accuracy over the study periods

B Appendix



(a) Bitcoin mean squared error over the different frequencies



(b) Mean squared error over the study periods

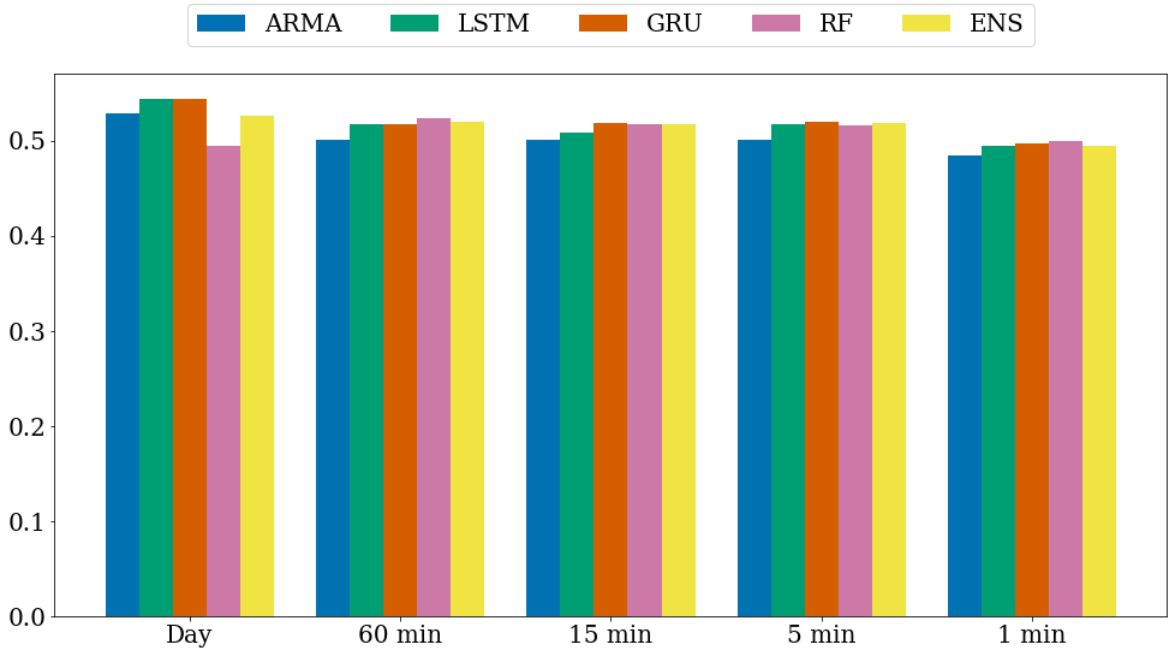
Figure B.1: Bitcoin mean squared error of ARMA, LSTM, GRU, RF and Ensemble model

Frequency	ARMA	LSTM	GRU	RF	ENS	Mean
1 day	2.08e-03	2.07e-03	2.09e-03	2.33e-03	2.10e-03	2.13e-03
60 minutes	1.15e-04	1.02e-04	1.02e-04	1.05e-04	1.02e-04	1.05e-04
15 minutes	3.07e-05	2.64e-05	2.65e-05	2.72e-05	2.66e-05	2.75e-05
5 minutes	1.19e-05	9.06e-06	9.07e-06	9.23e-06	9.22e-06	9.69e-06
1 minute	2.47e-06	1.48e-06	1.48e-06	1.48e-06	1.54e-06	1.69e-06
Mean	4.48e-04	4.41e-04	4.45e-04	4.94e-04	4.48e-04	4.55e-04

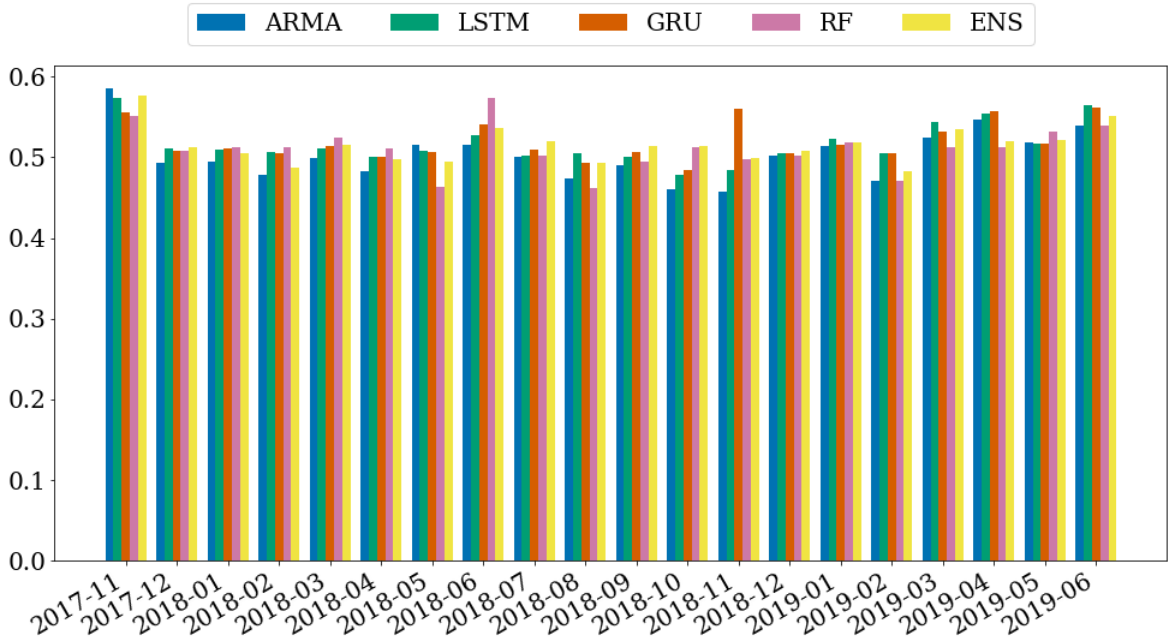
Table B.1: Bitcoin MSE over the frequencies

Study period	ARMA	LSTM	GRU	RF	ENS	Mean
2017-11	9.11e-04	9.31e-04	8.92e-04	9.45e-04	9.10e-04	9.18e-04
2017-12	9.80e-04	9.70e-04	9.80e-04	1.06e-03	9.83e-04	9.94e-04
2018-01	1.41e-03	1.35e-03	1.39e-03	1.44e-03	1.38e-03	1.39e-03
2018-02	6.65e-04	6.46e-04	6.46e-04	7.95e-04	6.60e-04	6.82e-04
2018-03	5.27e-04	5.25e-04	5.24e-04	6.14e-04	5.43e-04	5.46e-04
2018-04	4.38e-04	4.26e-04	4.60e-04	4.31e-04	4.34e-04	4.38e-04
2018-05	1.97e-04	1.92e-04	1.98e-04	2.34e-04	2.03e-04	2.05e-04
2018-06	2.39e-04	2.45e-04	2.33e-04	1.92e-04	2.22e-04	2.26e-04
2018-07	2.86e-04	2.86e-04	2.85e-04	3.24e-04	2.92e-04	2.94e-04
2018-08	1.58e-04	1.51e-04	1.56e-04	1.74e-04	1.56e-04	1.59e-04
2018-09	7.23e-05	7.23e-05	7.29e-05	9.19e-05	7.39e-05	7.67e-05
2018-10	2.25e-05	2.16e-05	2.33e-05	2.49e-05	2.23e-05	2.29e-05
2018-11	7.44e-04	7.44e-04	7.40e-04	7.84e-04	7.51e-04	7.53e-04
2018-12	4.45e-04	4.45e-04	4.57e-04	6.85e-04	4.81e-04	5.03e-04
2019-01	9.96e-05	9.83e-05	1.18e-04	1.10e-04	1.01e-04	1.05e-04
2019-02	1.43e-04	1.26e-04	1.25e-04	1.33e-04	1.28e-04	1.31e-04
2019-03	2.74e-04	2.55e-04	2.65e-04	2.59e-04	2.60e-04	2.62e-04
2019-04	2.87e-04	2.81e-04	2.84e-04	4.38e-04	3.09e-04	3.20e-04
2019-05	3.71e-04	3.77e-04	3.77e-04	3.95e-04	3.73e-04	3.79e-04
2019-06	6.91e-04	6.82e-04	6.83e-04	7.55e-04	6.90e-04	7.00e-04
Mean	4.48e-04	4.41e-04	4.45e-04	4.94e-04	4.48e-04	4.55e-04

Table B.2: Bitcoin MSE over the study periods



(a) Bitcoin directional accuracy over the different frequencies



(b) Directional accuracy over the study periods

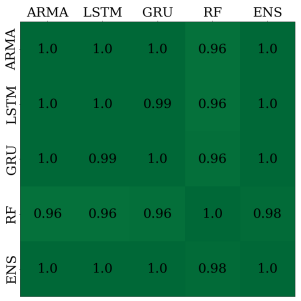
Figure B.2: Bitcoin directional accuracy of ARMA, LSTM, GRU, RF and Ensemble model

Frequency	ARMA	LSTM	GRU	RF	ENS	Mean
1 day	0.53	0.54	0.54	0.50	0.53	0.53
60 minutes	0.50	0.52	0.52	0.52	0.52	0.52
15 minutes	0.50	0.51	0.52	0.52	0.52	0.51
5 minutes	0.50	0.52	0.52	0.52	0.52	0.51
1 minute	0.48	0.50	0.50	0.50	0.49	0.49
Mean	0.50	0.52	0.52	0.51	0.52	0.51

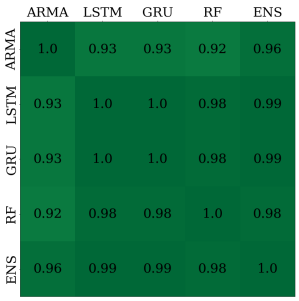
Table B.3: Bitcoin directional accuracy over the frequencies

Study period	ARMA	LSTM	GRU	RF	ENS	Mean
2017-11	0.58	0.57	0.56	0.55	0.58	0.57
2017-12	0.49	0.51	0.51	0.51	0.51	0.51
2018-01	0.50	0.51	0.51	0.51	0.51	0.51
2018-02	0.48	0.51	0.50	0.51	0.49	0.50
2018-03	0.50	0.51	0.51	0.53	0.51	0.51
2018-04	0.48	0.50	0.50	0.51	0.50	0.50
2018-05	0.52	0.51	0.51	0.46	0.50	0.50
2018-06	0.52	0.53	0.54	0.57	0.54	0.54
2018-07	0.50	0.50	0.51	0.50	0.52	0.51
2018-08	0.47	0.50	0.49	0.46	0.49	0.49
2018-09	0.49	0.50	0.51	0.49	0.51	0.50
2018-10	0.46	0.48	0.48	0.51	0.51	0.49
2018-11	0.46	0.48	0.56	0.50	0.50	0.50
2018-12	0.50	0.51	0.50	0.50	0.51	0.50
2019-01	0.51	0.52	0.52	0.52	0.52	0.52
2019-02	0.47	0.51	0.51	0.47	0.48	0.49
2019-03	0.52	0.54	0.53	0.51	0.54	0.53
2019-04	0.55	0.55	0.56	0.51	0.52	0.54
2019-05	0.52	0.52	0.52	0.53	0.52	0.52
2019-06	0.54	0.56	0.56	0.54	0.55	0.55
Mean	0.50	0.52	0.52	0.51	0.52	0.51

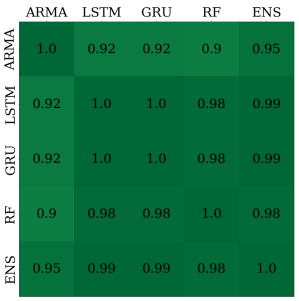
Table B.4: Bitcoin directional accuracy over the study periods



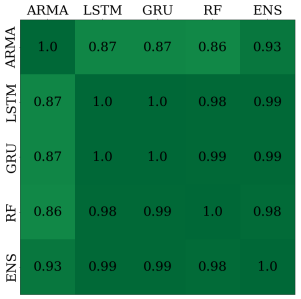
(a) Daily frequency



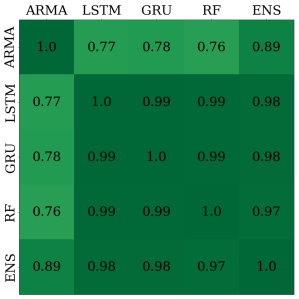
(b) 60 minutes frequency



(c) 15 minutes frequency

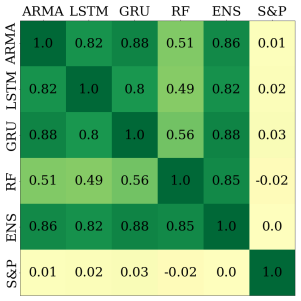


(d) 5 minutes frequency

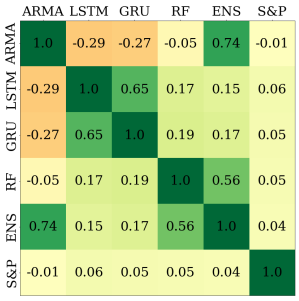


(e) 1 minute frequency

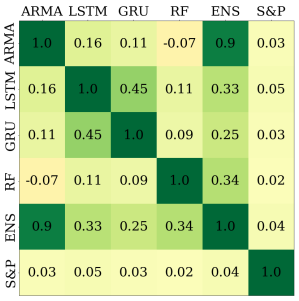
Figure B.3: Bitcoin correlation matrix of model errors at different frequencies



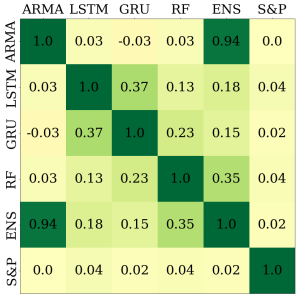
(a) Daily frequency



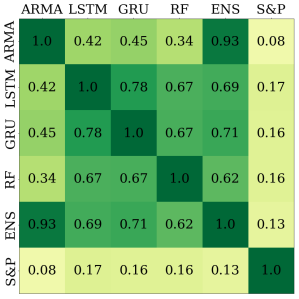
(b) 60 minutes frequency



(c) 15 minutes frequency



(d) 5 minutes frequency



(e) 1 minute frequency

Figure B.4: Correlation matrix of model predictions and Bitcoin at different frequencies

Frequency	ARMA	LSTM	GRU	RF	ENS	Bitcoin
1 day	6.66e-03	5.88e-03	8.12e-03	1.58e-02	7.79e-03	4.52e-02
60 minutes	3.56e-03	7.73e-04	7.32e-04	2.18e-03	1.02e-03	1.01e-02
15 minutes	2.18e-03	2.44e-04	1.81e-04	9.41e-04	6.03e-04	5.15e-03
5 minutes	1.68e-03	1.55e-04	1.43e-04	5.28e-04	4.52e-04	3.01e-03
1 minute	1.07e-03	2.17e-04	2.19e-04	2.46e-04	3.63e-04	1.23e-03

Table B.5: Standard deviation of model predictions and Bitcoin logarithmic returns.

Frequency	ARMA	LSTM	GRU	RF	ENS
1 day	0.3830	1.0000	0.3830	0.0000	0.1090
60 minutes	0.0530	1.0000	0.3270	0.0010	0.2760
15 minutes	0.0030	1.0000	0.1010	0.0050	0.0340
5 minutes	0.0000	1.0000	0.0100	0.0000	0.0000
1 minute	0.0000	1.0000	0.3150	0.3150	0.0000

Table B.6: Bitcoin model confidence set

Frequency	ARMA	LSTM	GRU	RF	ENS	Bitcoin
1 day	0.36	0.99	0.99	-0.77	0.72	0.76
60 minutes	-0.23	2.64	1.87	2.30	2.50	0.50
15 minutes	-0.23	2.64	1.87	2.30	2.50	0.50
5 minutes	0.40	8.17	5.66	8.22	6.95	0.50
1 minute	24.22	66.84	71.24	75.38	54.79	0.50

Table B.7: Bitcoin sum logarithmic return with no transaction cost

Frequency	ARMA	LSTM	GRU	RF	ENS	Bitcoin
1 day	1.49	2.16	1.67	-0.64	0.67	0.75
60 minutes	-3.67	0.41	-1.19	-1.26	1.41	0.49
15 minutes	-3.92	-0.69	-1.68	-4.45	-1.81	0.49
5 minutes	-17.70	0.95	-0.69	-3.57	-6.86	0.49
1 minute	-49.10	9.09	7.90	5.17	4.18	0.50

Table B.8: Bitcoin sum logarithmic return with 5 base points of transaction cost

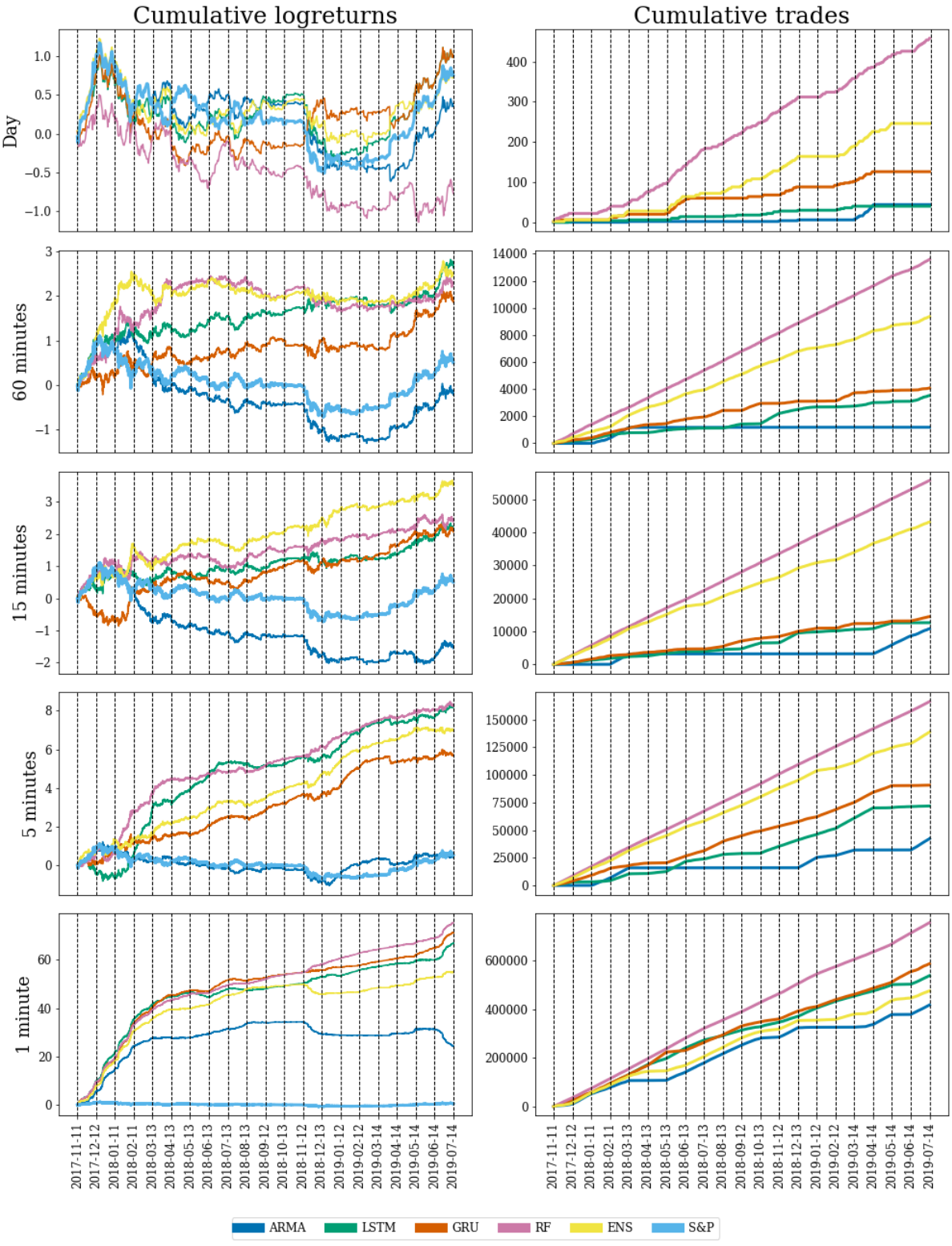


Figure B.5: Cumulative logarithmic return and cumulative sum of trades of ARMA, LSTM, GRU, RF, Ensemble model and Bitcoin at different frequencies with no transaction cost

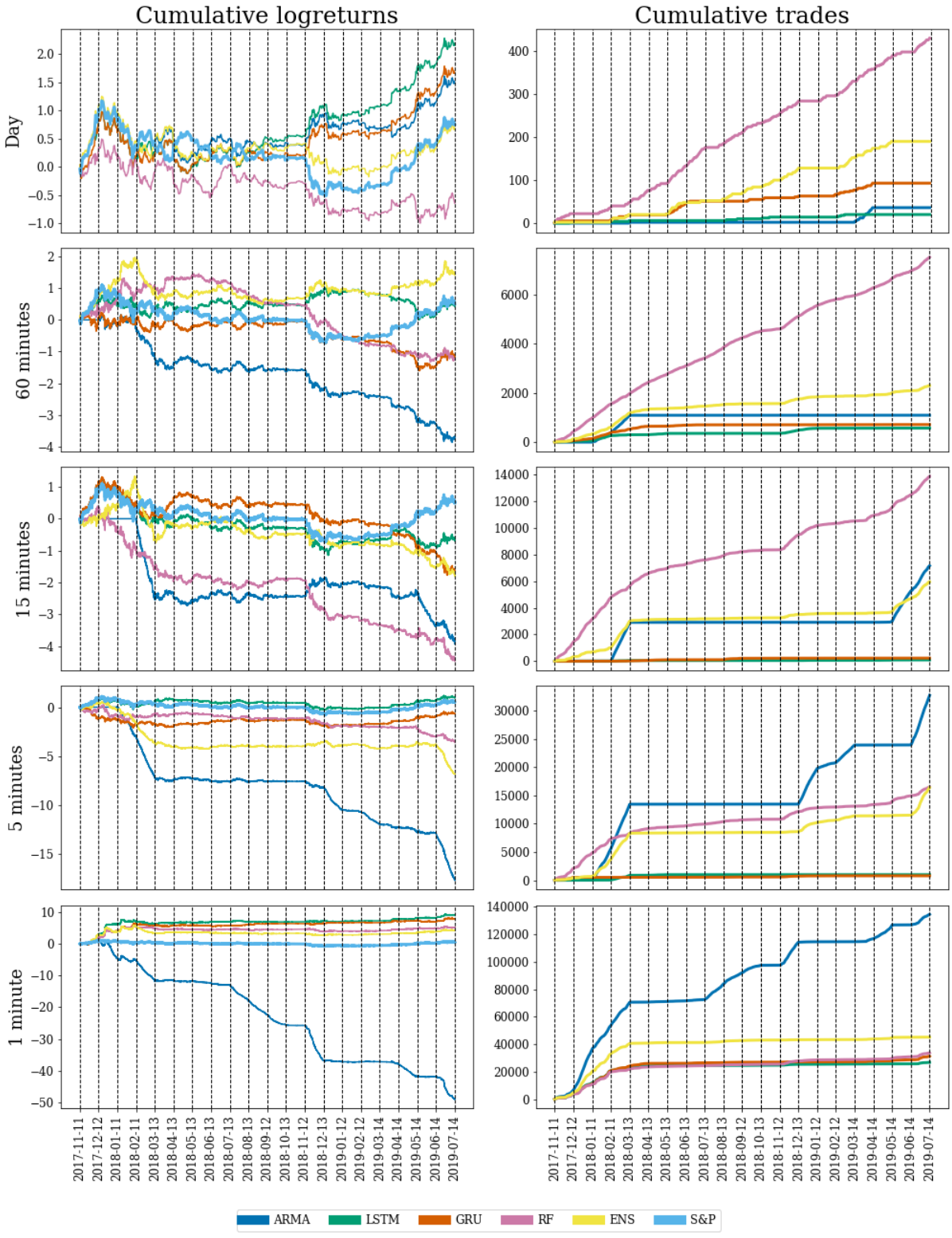


Figure B.6: Cumulative logarithmic return and cumulative sum of trades of ARMA, LSTM, GRU, RF, Ensemble model and Bitcoin at different frequencies with 5 base points of transaction cost

Frequency	ARMA	LSTM	GRU	RF	ENS	Bitcoin
1 day	0.28	0.45	0.45	-0.03	0.38	0.39
60 minutes	0.13	0.88	0.68	0.79	0.84	0.32
15 minutes	-0.20	0.74	0.72	0.82	1.09	0.32
5 minutes	0.30	2.25	1.62	2.26	1.94	0.32
1 minute	6.80	18.48	19.69	20.83	15.17	0.32

Table B.9: Bitcoin annualized sharpe ratio with no transaction cost

Frequency	ARMA	LSTM	GRU	RF	ENS	Bitcoin
1 day	0.59	0.78	0.64	0.00	0.36	0.38
60 minutes	-0.82	0.30	-0.11	-0.13	0.55	0.32
15 minutes	-1.20	0.01	-0.23	-0.93	-0.26	0.32
5 minutes	-5.04	0.44	0.02	-0.69	-1.51	0.32
1 minute	-13.00	2.68	2.35	1.60	1.32	0.32

Table B.10: Bitcoin annualized sharpe ratio with 5 base points of transaction cost

Frequency	ARMA	LSTM	GRU	RF	ENS	Bitcoin
1 day	0.39	0.64	0.65	-0.05	0.54	0.56
60 minutes	0.19	1.23	0.95	1.11	1.19	0.46
15 minutes	-0.28	1.03	0.99	1.15	1.52	0.45
5 minutes	0.42	3.11	2.23	3.21	2.73	0.46
1 minute	9.60	28.12	30.17	31.92	22.56	0.45

Table B.11: Bitcoin annualized sortino ratio with no transaction cost

Frequency	ARMA	LSTM	GRU	RF	ENS	Bitcoin
1 day	0.86	1.14	0.94	0.00	0.51	0.56
60 minutes	-1.15	0.42	-0.16	-0.18	0.80	0.46
15 minutes	-1.64	0.02	-0.32	-1.31	-0.37	0.45
5 minutes	-6.99	0.62	0.03	-0.98	-2.14	0.45
1 minute	-17.60	3.89	3.41	2.33	1.91	0.44

Table B.12: Bitcoin annualized sortino ratio with 5 base points of transaction cost