Stockholms universitet

Forecasting Univariate Financial Time Series using eXtreme Gradient Boosting

Adrian Axelsson

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2021:1
Matematisk statistik
Januari 2021

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Bachelor Thesis **2021:1**
http://www.math.su.se

Forecasting Univariate Financial Time Series using eXtreme Gradient Boosting

Adrian Axelsson*

January 2021

Abstract

The field of time series forecasting has for many decades been highly influenced by linear statistical models, but machine learning techniques have drawn much attention in this field in more recent times. The application of machine learning methods is not theoretically straightforward due to inherent serial correlation, although many practitioners seem to put less emphasis on this fact. This thesis aims to examine existing academic work on the formalization of univariate time series forecasting as a supervised learning task, and evaluate whether the currently popular eXtreme Gradient Boosting or XGBoost method, which has drawn much attention since its introduction in 2016 due to great success in various machine learning competitions, can be considered a potential method for the problem at hand. To this end, we give a fundamental background in traditional time series analysis, with particular emphasis on applications within finance, supervised learning and XGBoost in particular. We also present a theoretically justified supervised learning setting to one-step ahead forecasting of univariate time series. The theory is then applied in a forecasting case study on Nasdaq OMX Stockholm Large Cap Price Index daily returns, where performance of XGBoost and the traditional ARMA model is compared. We conclude that the formalization of one step-ahead forecasting of time series as a supervised learning task is justified. Furthermore, although weak forecasting results due to that the efficient market hypothesis seems to be applicable on the dataset, we conclude that XGBoost can capture the existing weak dynamic dependencies of data, and that this method itself possesses properties that could be beneficial in time series forecasting.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: axelsson.adrian@gmail.com. Supervisor: Ola Hössjer, Taras Bodnar, Pieter Trapman.

Acknowledgements

This is a Bachelor's thesis of 15 ECTS in Mathematical Statistics at the Department of Mathematics at Stockholm University. I would like to thank my supervisors Ola Hössjer, Taras Bodnar and Peter Trapman for their support and advice during the writing of this thesis. I also like to thank my family for their support.

Contents

1 Introduction

1.1 Background

Time series arise when events can be observed sequentially in time. Analysis of such series is of great interest in many fields, not the least in finance and economics, where stochastic and dynamic models are applied to describe the dependency structure between adjacent observations. These models are used for several purposes, including forecasting future values of a time series from current and past values.

Traditional linear time series theory has highly influenced the field of forecasting for many decades, but in more recent times, machine learning methods have caught the attention of many practitioners. One of the earliest contestants to the traditional models was Artificial Neural Network (ANN). In this thesis we will adopt a supervised learning setting to forecasting of univariate financial time series, influenced primarily Bontempi et al. (2003) [1], and Ahmed et al. (2010) [2]. We implement the currently popular eXtreme Gradient Boosting (XGBoost) method by Chen & Guestrin (2016) [3], which has achieved great success in various machine learning competitions considering both classification and regression problems.

1.2 Aim

The main aim of this thesis is to examine whether the supervised learned setting to time series forecasting, used in above papers, is theoretically justifiable, and if XGBoost could be a potential method to be applied in this setting. We focus specifically on forecasting of financial time series. The intended reader is assumed to possess knowledge corresponding to the bachelor's level in mathematical statistics.

1.3 Methodology

We use traditional linear financial time series theory as a starting point of the forecasting task in this thesis, for univariate financial time series in particular. This theory provides some fundamental knowledge on distinctive characteristics of financial time series and how traditional stochastic models are applied to forecast such series. The weak stationary property of the time series at hand is central in this theory, and has great influence on the choice of pre-processing methods and statistical tests performed on data in a case study, introduced below.

We also provide a fundamental background in supervised learning as function approximation, especially in regression problems. This includes, among other things, cross-validation methods for model selection and assessment, which is an integral part of the supervised learning setting to time series forecasting. A theoretical derivation of XGBoost in a regression setting is also given, which we derive as an extension of Newton Boosting (Sigrist, 2019) [4].

The findings in the above sections are then implemented in a case study, which aims to evaluate the predictive performance of XGBoost on real world data. The dataset was determined in advance, and consists of Nasdaq OMX Stockholm Large Cap Price Index (OMXSLCPI) daily returns during the period 2015-2019. The well known linear ARMA model is used as benchmark model in order to evaluate the performance of XGBoost.

1.4 Main Results

We find that academic work that supports forecasting of univariate time series as a supervised learning task exist. The introduced supervised learning setting is justified using the findings in Bergmeir et al. (2017) [5], which assesses the validity of k-fold cross-validation for evaluating autoregressive time series forecasting models, where machine learning methods are included.

The result of the case study shows that neither XGBoost nor ARMA perform better than a mean model, which we argue follows from that the efficient market hypothesis seems to hold for OMXSLCPI daily log returns during the current period. However, from observing feature importance we find that XGBoost can capture existing weak dynamic dependencies in data, and that the method itself possesses properties that could be beneficial in time series forecasting.

1.5 Disposition

Some fundamental theory of financial time series forecasting is given in section 2. We then proceed with basic supervised learning theory in section 3, which ends with the supervised learning setting to time series forecasting of univariate time series used in this thesis. The case study is described in section 4, and finally in section 5 the findings in this thesis are discussed, conclusions are stated and some suggestions for further studies are proposed.

1.6 Literature

The theory on financial time series is almost exclusively based on *Analysis of Financial Time Series* by Tsay (2010) [6], which offers comprehensive theoretical and empirical content for modelling of financial time series. *Time Series Analysis: Forecasting and Control* by Box et al. (2015) [7], has been used as supplementary literature. Primarily since this book treats time series in general and not specifically financial time series.

The general theory on statistical learning is mainly based on *Elements of Statistical Learning* by Hastie et al. (2009) [8], and to some extent from *Machine Learning: A Probabilistic Perspective* by Murphy (2012) [9]. The authors of these two books come from different fields, but both adopt a probabilistic approach to statistical learning or machine learning. We found that these books complement each other well, as simultaneous reading facilitated understanding of various related concepts. The derivation of eXtreme Gradient Boosting is based on *XGBoost: A Scaleable Tree Boosting System* by Chen & Guestrin

(2016) [3], but also the papers by Friedman et al. (2000) and Friedman (2001, 2002) [10, 11] provided a necessary background for gradient tree boosting methods.

The papers by Bontempi et al. (2013) [1], Ahmed et al. (2010) [2] and Bergmeir et al. (2017) [5] form the basis for the supervised learning setting to time series forecasting.

2 Time Series Analysis

In this section we will go through some basic concepts of univariate financial time series analysis and forecasting. We derive the traditional linear time series model ARMA, which is used as benchmark model in section 4. A basic understanding of the concepts in this section will also facilitate comprehending the implementation of supervised learning models on this type of problem, which we do in section 3.1.

This section is almost exclusively based on *Analysis of Financial Time Series* by Tsay (2010) [6], and *Time Series Analysis: Forecasting and Control* by Box et al. (2015) [7].

2.1 Time Series

2.1.1 Time Series as a Stochastic Process

We define a time series $\{z_t\}_1^N = z_1, z_2, ..., z_N$ as a set of successive realization of a stochastic process, where the observations are taken on equally spaced time intervals. Typically, such a series possess the property that adjacent observations are dependent. Time series analysis concerns analysing this dependence, which we in this thesis will do with the purpose of forecasting. Many time series not only exhibit dependence between adjacent observations, but between observations of multiples of lags. Such series are said to possess seasonal or cyclic dependence, determined on the size of the multiple.

2.1.1.1 Forecasting

Time series forecasting use the at a current time t available observation z_t and passed or lagged observations $z_{t-l}, z_{t-2}, ..., z_{t-(N-1)}$ from a time series, in order to build a method which can predict future or lead values $z_{t+1}, z_{t+2}, ..., z_{t+h}$, for some h. The case of $h > 1$ is referred to as multi step-ahead forecasting, whilst the case of $h = 1$ is referred to as one step-ahead forecasting. We will concern the latter and denote the forecast estimate by $\hat{z}_t(1)$. The objective is to obtain a model for which the expected square deviation given the previous information $E[(z_{t+1} - \hat{z}_t(1))^2 | z_t, z_{t-1}, ..]$ becomes as small as possible.

2.1.2 Stationary Stochastic Process

The special class of stationary stochastic processes is central in time series analysis, and linear time series models rely on such assumption. In this subsection we will define two types of stationary processes.

Let $\{z_t\}$ be a time series, and let $F_Z(z_{t_1}, z_{t_2}, ..., z_{t_m})$ be the cumulative distribution function associated with m observations, made at any set of times $t_1, t_2, ..., t_m$. Then $\{z_t\}$ is said to be *strictly stationary* if

$$F_Z(z_{t_1}, z_{t_2}, ..., z_{t_m}) = F_Z(z_{t_1+k}, z_{t_2+k}, ..., z_{t_m+k})$$

for any positive discrete time shift k, for any m, and for any $t_n, ..., t_m$.

When $m = 1$, the strict stationary assumption implies that $F_Z(z_t) = F_Z(z_{t+k})$. This means that the probability distribution is the same for all t, and the mean and variance of the time series $\{z_t\}$ would be constant, when they exist.

The assumption of strict stationarity is not realistic on empirical data. For prediction purposes, one often settles for weak stationarity.

Let $\{z_t\}$ be a time series, which is said to be *weakly stationary* if

$$\begin{array}{lll} (a) & E[z_t] & = \mu \\ (b) & Cov(z_t, z_{t-l}) & = \gamma_l, \end{array}$$

for any l and t. This means that a process is weakly stationary if the first two moments of z_t are the same for all t, and that the covariance between z_t and z_{t-l} only depends on the lag size l.

A special case is when $\{z_t\}$ is a Gaussian process, then weak stationarity is equivalent to strict stationarity. Strict stationarity does in general not imply weak stationarity, since finite second moment is not assumed for strict stationary process.

2.1.3 The Autocorrelation Function

To measure the linear relationship between z_t and past returns, we use a concept called *autocorrelation*, which is a generalization of correlation. The correlation between z_t and z_{t-l} is called the lag-l *autocorrelation*, which for a weak stationary time series $\{z_t\}$ is defined by:

$$\rho_l = \frac{Cov(z_t, z_{t-l})}{\sqrt{Var(z_t)Var(z_{t-l})}} = \frac{Cov(z_t, z_{t-l})}{Var(z_t)},$$

where we have used that $Var(z_t) = Var(z_{t-l})$ under weak stationary. For a given sample $\{z_t\}_{i=1}^N$, we estimate ρ_l using the the lag-l *sample autocorrelation*

$$\hat{\rho}_l = \frac{\sum_{t=l+1}^{N}(z_t - \bar{z}))(z_{t-l} - \bar{z})}{\sum_{t=1}^{T}(z_t - \bar{z})^2}, \quad 0 \leq l < N - 1,$$

where $\bar{z}$ is the sample mean. The series $\hat{\rho}_1, \hat{\rho}_2, ...$ is called the *sample autocorrelation function* (ACF) of z_t.

In the context of forecasting, we assume that past observations give us significant information about the behaviour of future observations, that is, ρ_l is significantly different from zero, for some $l > 0$. The *Ljung-Box test*, Ljung & Box (1978) [12], is commonly used to test the null hypothesis $H_0 : \rho_1 = \rho_2 = ... = \rho_m = 0$, against the alternative hypothesis $H_1 : \rho_i \neq 0$, for some $i \in \{1, 2, ..., m\}$. Under H_0, the test statistic

$$Q(m) = N(N+2)\sum_{l=1}^{m}\frac{\hat{\rho}_l^2}{N-l} \tag{1}$$

follows asymptotically a χ^2 distribution with m degrees of freedom. We will reject H_0 in favor of H_1 if the p-value corresponding to $Q(m)$ is less than our significance level. Tsay (2010, pp. 33) [6] suggests using $m \approx \ln(N)$.

The Ljung-Box test can also be applied to assess fully specified ARMA models (Tsay, 2010, pp. 68) [6]. In such application, we test the model residuals for significant autocorrelations. If the model successfully captures the dynamic dependency, the residuals should show no significant autocorrelations. In this case, under H_0, the test statistic $Q(m)$ in (1) should follow asymptotically a χ^2 distribution with $m - p - q$ degrees of freedom, where p and q define the order of the model (see section 2.3.3).

A related concept is the partial autocorrelation, which we will briefly present. The interested reader is referred to Tsay (2010, sect. 2.4.2) [6] or Box et al. (2015, sect. 3.2.5) [7] for more details. The lag-l *sample partial autocorrelation* (PACF) measures the contribution of adding another lagged observation to an autoregressive model, and can thus be used for determining the order p of the model.

2.2 Financial Time Series

Having gone through some basic concepts of time series in general, we will from now on focus on a certain type, namely financial time series. Studies in financial time series mainly concern returns, rather than prices (Campbell, 1997) [13]. There are two main reasons behind this. Firstly, the return is a complete and scale-free summary of the investment opportunity. This means that under the assumption of perfect competition on the market, the market share has no influence on prices, thus the return is a useful measure for the general investor. Secondly, returns have some appealing statistical properties related to their behaviour over time. The concepts derived in this chapter mainly concern return series.

We begin this section by defining asset return series and derive some assumptions regarding their distributional properties and modeling. This content is almost exclusively based on *Analysis of Financial Time Series* (Tsay, 2010) [6].

2.2.1 One-Period Returns

There are several definitions of asset returns, those treated in this thesis are defined below.

Let P_t be the price of an asset, that pay no dividend, at time t. Holding the asset for one period from date $t-1$ to date t would result in a *simple gross return:*

$$1 + R_t = \frac{P_t}{P_{t-1}} \quad \Longleftrightarrow \quad P_t = P_{t-1}(1 + R_t),$$

where

$$R_t = \frac{P_t}{P_{t-1}} - 1 = \frac{P_t - P_{t-1}}{P_{t-1}}$$

is the *simple return*. Thus simple return is the percentage gain for holding an asset during one period of time. Taking the natural logarithm of the *simple gross return* $1+R_t$ proves useful in analysis of financial time series, and is called *log return:*

$$r_t = \ln(1 + R_t) = \ln(\frac{P_t}{P_{t-1}}).$$

Note that log returns are additive, meaning we can calculate the total return over k periods by simply adding the k individual one step log returns.

2.2.2 The distributions of returns

With the objective to forecast log returns for individual assets, we shall specify a model that makes use of an assumed dynamic dependence to describe the stochastic behaviour of a return series $\{r_t\}_1^T$. The starting point for such model is the joint distribution function of T succeeding stochastic returns for which the joint distribution function is uniquely determined by the unknown parameter θ:

$$F(r_1, r_2, ..., r_T; \theta). \tag{2}$$

To study the dynamic structure of asset returns it is useful to partition the joint distribution (2) (Tsay, 2010, pp. 15) [6] as

$$F(r_1, r_2, ..., r_T; \theta) = F(r_1; \theta) \prod_{t=2}^{T} F(r_t | r_{t-1}, ..., r_1; \theta), \tag{3}$$

which shift our focus to the conditional distribution $F(r_t|r_{t-1}, ..., r_1; \theta)$. This conditional distribution must be chosen appropriately, such that it sufficiently describes the asset return over time. Assuming such conditional distribution is specified, it remains to estimate θ in order to make inference about asset returns, using for example maximum-likelihood estimation.

A special case related to forecasting is when the conditional distribution in (3) is equal to the marginal distribution $F(r_t)$. In this case, there is no dynamic dependence between asset returns. The forecast of such series is the naive one, namely the sample mean.

2.2.2.1 Normal Assumption of Returns

It is common to assume that log returns r_t are independent and identically distributed random variables following a normal distribution with mean μ and variance σ^2. However, in empirical studies of asset returns the normal assumption is often questionable. Typically the distribution is asymmetric (skew) and have heavy tails (excess kurtosis). Asymptotic properties of the sample counterparts to the skewness statistic $S(r_t) = E[(r_t - \mu)^3/\sigma^3]$ and the kurtosis statistic $K(r_t) = E[(r_t - \mu)^4/\sigma^4]$ can be used to test the normal assumption of data (Tsay, 2010, pp. 9) [6].

Let $\hat{\mu}$ and $\hat{\sigma}^2$ be the sample mean and variance respectively of $\{r_t\}_1^T$. Then

$$\hat{S}(r_t) = \frac{1}{(T-1)\hat{\sigma}^3} \sum_{t=1}^{T} (r_t - \hat{\mu})^3,$$
$$\hat{K}(r_t) = \frac{1}{(T-1)\hat{\sigma}^4} \sum_{t=1}^{T} (r_t - \hat{\mu})^4,$$

are the sample skewness and kurtosis respectively. $\hat{S}(r_t)$ and $\hat{K}(r_t) - 3$ follow asymptotically a normal distribution with zero mean, and variance $6/T$ and $24/T$ respectively. This means that the test statistics

$$t_s = \frac{\hat{S}(r_t)}{\sqrt{(6/T)}},$$
$$t_k = \frac{\hat{K}(r_t) - 3}{\sqrt{(24/T)}},$$

are standard normal distributed. The respective null hypothesis $H_0 : S(r_t) = 0$ and $H_0 : K(r_t) - 3 = 0$ are rejected in favour of the alternative hypothesis $H_1 : S(r_t) \neq 0$ and $H_1 : K(r_t) - 3 \neq 0$, if the p-value corresponding to $|t_s|$ or $|t_k|$ is less than the chosen significance level.

2.2.2.2 Likelihood Function of Log Returns

Under the normal assumption of log returns the likelihood function of the log return series $\{r_t\}_1^T$ is

$$\mathcal{L}(r_1, r_2, ..., r_T; \theta) = F(r_1; \theta) \prod_{t=2}^{T} \frac{1}{\sqrt{2\pi\sigma_t(\theta)}} \exp\left[\frac{-(r_t - \mu_t(\theta))^2}{2\sigma_t^2(\theta)}\right]$$

where we have used the partitioned represenation of the distribution function in equation 3. The θ that maximises this expression is called the maximum-likelihood estimation (MLE).

2.3 Linear Time Series Models

We continue to consider succeeding stochastic asset returns r_t, which constitute a time series $\{r_t\}_1^T$, and wish to model its dynamic structure in order to forecast future returns. In this thesis we only concern possible traditional time series models that make a strong assumption of linear relationship between the returns. The background to this restriction is the purpose of using such model as a benchmark, when assessing the performance of XGBoost in the case study in section 4.

The considered model is the *autoregressive moving average* model (ARMA), which is a composition of the *autoregressive* and the *moving average* model.

This model assumes a weak stationary time series, which generally holds for a log return series. Price series however, do naturally appear to be non-stationary. We will briefly make a connection between modeling of these two types of financial time series by generalizing ARMA to the *autoregressive integrated moving average* model (ARIMA), which is used for modeling non-stationary time series, such as price series.

2.3.1 White Noise

The basis of many stochastic time series models is that successive observations are generated from a series $\{a_t\}$ of independent and identically distributed shocks a_t, having finite mean and variance, which are drawings from a fixed distribution. Such series is called a white noise series (Box et al., 2015, pp. 7) [7]. If a_t is normally distributed with mean zero and variance σ_a^2, the series is called a Gaussian white noise series. In practice, if a time series has all sample ACFs close to zero, then the series is approximately a white noise series (Tsay, 2010, pp. 36) [6].

2.3.2 The General Linear Time Series Model

A time series r_t is said to be linear if it can be written on the form

$$r_t = \mu + \sum_{i=0}^{\infty} \psi_i a_{t-i}, \tag{4}$$

where μ is the mean of r_t, $\psi_0 = 1$, and $\{a_t\}$ is a white noise series. The white noise variable can be considered the new information at time t, and its influence on the value of r_t is controlled by the *weight* parameter ψ.

Assuming that r_t is on the form (4) and that it possesses the weak stationary property, we can derive some of its properties using basic probability theory. First, we have $E(r_t) = \mu$ and $Var(r_t) = \sigma_a^2 \sum_{i=0}^{\infty} \psi_i^2$. Furthermore, from the derivation of lag-l *autocovariance* of r_t, we can conclude that the weights are related to the autocorrelations of r_t and that remote returns have little or no influence on current return r_t. For details, see Tsay (2010, sect. 2.3) [6].

2.3.3 The ARMA Model

The ARMA model has been used for analysis of financial time series for decades, but more so in the past than in the present. This model is the result of combining two simpler linear models in order to overcome possible high-order issues occurring when they are applied separately. These models are the *autoregressive* model, AR(p), and the *moving-average* model, MA(q), which in short are focused on modelling the autocorrelation parameters and the white noise series respectively. These models can also be represented as special cases of the ARMA model.

The general ARMA(p,q) model is on the form

$$r_t = \phi_0 + \sum_{i=1}^{p} \phi_i r_{t-i} + a_t - \sum_{i=1}^{q} \theta_i a_{t-i}, \tag{5}$$

where $\{a_t\}$ is a white noise series, and p and q are non-negative integers. In this thesis we assume $\{a_t\}$ is a Gaussian white noise series. It turns out that r_t which satisfies equation (5) is weakly stationary if all the solutions to the associated *charasteristic equation*

$$1 - \phi_1 x - \phi_2 x^2 - \cdots - \psi_p x^p = 0 \tag{6}$$

are less than one in absolute value. The characteristic equation is related to the AR part, since the MA part is always stationary.

2.3.3.1 Order Determination

There are several approaches to selecting the parameters p and q of the ARMA model. Box et al. (2015, pp. 183) [7] primarily present methods related to characteristics of the sample ACF and PACF, while Tsay (2010, pp. 66) [6] primarily suggest using the *extended auto-correlation function* (EACF) (Tsay & Tiao, 1984) [14]. Among alternative methods they suggest using an information criteria, which assess a specified model according to a trade-off between goodness of fit and complexity penalization. We define three well known information criteria AIC (Akaike, 1973) [15], AICc (Sugiura, 1978) [16] and BIC (Schwartz, 1978) [17] following Burnham & Anderson (2002) [18].

Let $\mathcal{L}(\hat{\theta}_{\mathrm{ML}})$ be the value of the maximized model maximum-likelihood function, and let $g = p + q$, then

$$\begin{aligned} \mathrm{AIC} &= -2\ln(\mathcal{L}(\hat{\theta}_{\mathrm{ML}})) + 2g \\ \mathrm{AICc} &= \mathrm{AIC} + \tfrac{2g(g+1)}{T-g-1} \\ \mathrm{BIC} &= -2\ln(\mathcal{L}(\hat{\theta}_{\mathrm{ML}})) + g\ln(T) \end{aligned} .$$

In this thesis, we will use the ACF and PACF to estimate reasonable ranges $0 \leq p \leq P$ and $0 \leq q \leq Q$, then evaluate all possible combinations using an AICc in order to perform model selection.

2.3.3.2 Forecasting with ARMA

Let h denote the forecast origin and let F_h denote the by h available information. The *one-step-ahead* forecast r_{h+1} of the return series using ARMA is defined by

$$\hat{r}_h(1) = E[r_{h+1}|F_h] = \phi_0 + \sum_{i=1}^{p} \phi_i r_{h+1-i} - \sum_{i=1}^{q} \theta_i a_{h+1-i}, \tag{7}$$

with associated standard deviation $\sqrt{Var(e_h(1))} = \sqrt{Var(r_{h+1} - \hat{r}_h(1))} = \sqrt{Var(a_{h+1})} = \sigma_a$ of the forecast error $e_h(1)$.

2.3.4 Extending ARMA to ARIMA

If we allow the solutions to the characteristic equation (6) in section 2.3.3 to be equal to one, ARMA is extended to the ARIMA model. A time series that possesses this property is said to be *unit-root nonstationary*. A conventional approach to handle such series is to transform it to a stationary series using *differentiation*. Differentiation means transforming a time series z_t to c_t by letting $c_t = z_t - z_{t-1} = (1 - B)z_t$, where the *back shift operator* B is such that $Bz_t = z_{t-1}$.

This model comes in handy if we instead of the log return series r_t consider the asset log price series p_t, which in general appear to be non-stationary by nature and can be such that it has a unit-root. If differencing the price series p_t once and the obtained return series r_t follows and invertible ARMA(p,q) model, then p_t follows an ARIMA(p,1,q) model.

To test whether a financial time series has a unit root it is common to perform the *Augmented Dickey-Fuller* test, Tsay (2010, section 2.7.5) [6].

3 Supervised Learning Approach

Having derived some fundamental properties of financial time series and how these can be studied using the traditional linear model ARMA, we will now switch focus to our alternative approach on forecasting future values of univariate financial time series.

Starting with a general introduction to statistical learning and supervised learning, we then proceed to common learning methods which underlie the XGBoost model, which is the model of primary interest in this thesis. In the final section we derive the chosen approach to reframe the univariate time series forecasting problem into a supervised learning problem.

The general theory on statistical learning in sections 3.1-3.3 is mainly from *Elements of Statistical Learning* by Hastie et al. (2009) [8], and to some extent from *Machine Learning: A Probabilistic Perspective* by Murphy (2012) [9]. The derivation of eXtreme Gradient Boosting is based on *XGBoost: A Scaleable Tree Boosting System* by Chen & Guestrin (2016) [3].

3.1 Supervised Learning

This section mainly follows Hastie et al. (2009, section 2.6) [8], which define statistical learning as a statistical or applied mathematics approach to machine learning.

3.1.1 Function Approximation

Supervised learning is a predictive task, where we are given a random sample $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ of independent input-output pairs called a *training set*, which are considered to be realizations from a joint distribution $\Pr(\mathbf{X}, Y)$. The aim

is to make a useful approximation of a function $f(\mathbf{X})$ in an assumed additive relationship

$$Y = f(\mathbf{X}) + \varepsilon \tag{8}$$

between the response variable Y and the p-dimensional predictor variable $\mathbf{X} = (X_1, X_2, ..., X_p)^T$, where the random error ε has $E[\varepsilon] = 0$ and $Var(\varepsilon) = \sigma^2$, and is independent of $\mathbf{X}$. The error term is assumed to capture unmeasured contributions, such as measurement errors and other unknown influential predictors. The input data pairs $(\mathbf{x}_i, y_i)$, $i = 1, 2, ..., N$, are viewed as points in a $(p+1)-$dimensional Euclidean space.

In this thesis, we are concerned with the case where both input and output are *quantitative* or *continuous*, meaning the domain of $\mathbf{X}$ is $\mathbb{R}^p$ and possible outcomes of Y is in $\mathbb{R}$. This entails the prediction task to be regression.

3.2 Model Assessment and Selection

In this section we shall enhance our knowledge in the procedure of estimating the predictive performance of a given learning method. There are two aims of such estimation: *Model Selection*: estimating the performance of different models in order to choose the best one, and *Model Assessment*: having selected a final model, estimating its prediction error (generalization error) on new data (Hastie et al. 2009, page 222) [8].

We illustrate two common approaches for model selection and assessment in Figure 1, where the dataset is randomly partitioned into subsets referred to as training, validation and test sets. In (a) the model is fit to the training set, then model selection is performed by estimating predictive performance on the validation set. The selected model is then assessed by estimating its predictive performance on the test set. The partition in (b) is related to k-fold cross-validation, which we will derive in section 3.2.3.

3.2.1 Performance Measures

Assume that we have initially chosen a prediction model and learning method, and that we have obtained an approximation $\hat{f}(\mathbf{X})$ of $f(\mathbf{X})$ in (8) by fitting the model to the training set $\mathcal{T}$. In order to estimate its predictive performance on new data, we first introduce the *loss function* $L(Y, \hat{f}(\mathbf{X}))$, which penalizes prediction errors. In regression problems, the most common choice is the *squared error loss* $L(Y, \hat{f}(\mathbf{X})) = (Y - \hat{f}(\mathbf{X}))^2$.

To evaluate model performance we want to estimate the *test error* or *generalization error*

$$\mathrm{Err}_{\mathcal{T}} = E[L(Y, \hat{f}(\mathbf{X}))|\mathcal{T}]$$

which is the prediction error over and independent test sample, where $\mathbf{X}$ and Y are drawn from the joint distribution $Pr(\mathbf{X}, Y)$. Note that the training set $\mathcal{T}$ is

fixed, and the prediction error is thus specific to this training set. It is however often more convenient to estimate the *expected test error*

$$\text{Err} = E[L(Y, \hat{f}(\mathbf{X}))] = E[\text{Err}_{\mathcal{T}}], \tag{9}$$

which removes randomness from the training set $\mathcal{T}$ by averaging.

A candidate estimate of the test error $Err_{\mathcal{T}}$ would be the *training error*

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(\mathbf{x}_i)),$$

which is the average error on the training set $\mathcal{T}$. However, this quantity has two major drawbacks. Firstly, it is a too optimistic estimate of the test error $\text{Err}_{\mathcal{T}}$. This is since $\hat{f}$ is specified through a fitting procedure on the specific training set $\mathcal{T}$, and it is highly likely that it would perform better on this dataset compared to a new random sample. Secondly, the training error tends to zero as the complexity of the model increases which makes it inappropriate for model selection. Complex models tend to overfit data which has a negative affect on its performance on new data.

3.2.2 The Bias-Variance Trade-Off

Assume a model $Y = f(\mathbf{X}) + \varepsilon$ (8), where $E[\varepsilon] = 0$ and $Var(\varepsilon) = \sigma^2$. For squared error loss and a new random point $\mathbf{x}_0$ we can express the expected test error Err (9) as

$$\begin{aligned}
\text{Err}(\mathbf{x}_0) &= E[(Y - \hat{f}(\mathbf{x}_0))^2 | \mathbf{X} = \mathbf{x}_0] \\
&= \sigma^2 + (E[\hat{f}(\mathbf{x}_0)] - f(\mathbf{x}_0))^2 + (E[\hat{f}(\mathbf{x}_0) - E\hat{f}(\mathbf{x}_0)])^2 \\
&= \sigma^2 + \text{Bias}^2(\hat{f}(\mathbf{x}_0)) + Var(\hat{f}(\mathbf{x}_0)) \\
&= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance}
\end{aligned},$$

which is known as the *bias-variance decomposition*. The irreducible error derives from how the target Y varies around its true mean and can thus not be influenced by our model selection. However, both the squared bias $(E[\hat{f}(\mathbf{x}_0)] - f(\mathbf{x}_0))^2$ and the variance $E[(\hat{f}(\mathbf{x}_0) - E[\hat{f}(\mathbf{x}_0)])^2]$ depend on our selected model $\hat{f}$ which thus controls how these factors contribute to the expected prediction error. A simple model that makes strong structural assumptions, like linearity, typically has large bias but small variance, whilst a more complex model with mild structural assumptions has small bias but large variance. The decision on model complexity thus controls the influence of bias and variance on the expected prediction error, and involves a *bias-variance trade-off*. If a special structure exists, then this can be used to reduce both bias and variance of the estimates. Complexity constraints, are often built into the learning method and rely on assuming some type of regular behaviour for input points that are close to each other.

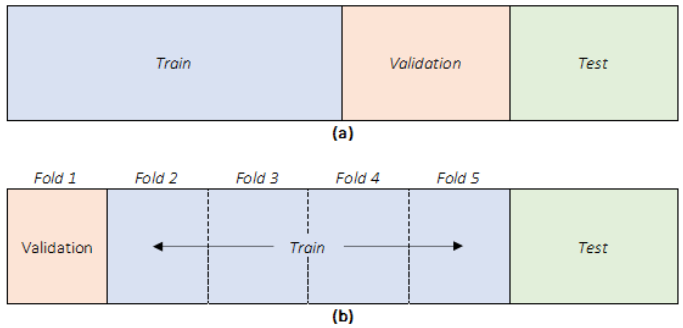

Figure 1: Illustration of partitioning data (a) into train, validation and test set (b) for 5-fold cross-validation

3.2.3 K-fold Cross-Validation Estimation of Err

K-fold cross-calidation is a popular method for estimating model performance, which attempts to directly estimate the expected prediction error Err. Instead of partitioning the available data into training and validation sets, k-fold cross-calidation partition data into k roughly equal-sized subsets, or folds. Err is then estimated on each fold by fitting the model to the remaining $k-1$ folds. The cross-calidation estimate of Err is the average of these k estimates, and is used as performance measure for model selection. The final model is then fit to the union of the k folds and model assessment is performed on the test set. Figure 1 (b) illustrates the partitioning in case of 5-fold cross-validation.

Now, we formally define k-fold cross-calidation, according Hastie et al. (2009, sect. 7.10.1) [8].

Let $\kappa : \{1, ..., N\} \mapsto \{1, ..., k\}$ be a function that randomly allocates the N observations in the dataset into k disjoint, roughly equal-sized subsets or *folds*. Further, let $\hat{f}^{-k}(\mathbf{x})$ denote the function fitted on the dataset having removed the kth subset. Then

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(\mathbf{x}_i)), \tag{10}$$

is the k-fold cross-validation estimate of the prediction error.

Given a set of models $f(\mathbf{x}; \alpha)$, where α is a tuning parameter, let $\hat{f}^{-k}(\mathbf{x}; \alpha))$ denote the αth model fit on the dataset having removed the kth subset. Then

$$CV(\hat{f}, \alpha) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(\mathbf{x}_i; \alpha)), \tag{11}$$

estimates the test error curve. We then choose $f(x, \hat{\alpha})$, where $\hat{\alpha}$ minimizes the test error curve, to be our chosen model. This model is then fit to the original dataset.

Choosing the number of folds k is a decision including a bias-variance trade-off. Choosing k large, means low bias but variance tend to be high since there will be few observations in each fold. On the other hand, small k leads to decrease in variance but increase of bias. In general, $k = 5$ or $k = 10$ have proven to be a good compromise.

3.3 Common Learning Methods

3.3.1 Linear Methods (for Regression)

In many fields of mathematics and statistics, linear methods are the starting point for more complex models. We will thus briefly describe linear methods in supervised learning.

The linear regression model is on the form

$$f(\mathbf{X}) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j, \tag{12}$$

where β_0 is the intercept. It is common to use least squares method to fit the linear model to the training set. This means finding $\hat{\beta}$ which minimizes the residual sum of squares

$$\begin{aligned} RSS(\beta) &= \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 \\ &= \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T \beta)^2, \end{aligned} \tag{13}$$

where we have set $x_{i,0} = 1$ for convenient estimation of the intercept $\beta_0 \in \mathbb{R}$.

An important property of the linear model is that it can be applied to transformations of the inputs, and can be presented as a special case of a broader class of linear model called *linear basis expansion* methods, which widely expands the applications of linear models.

3.3.2 Basis Function Expansions

One approach to move beyond linearity is by transforming the input variables in (12) using *basis functions*, which increase and/or replace the input variables. We follow Hastie et al. (2009, sect. 5.1) when defining the linear basis expansion in $\mathbf{X}$. Note that the intercept is dropped. The interested reader is also referred to Murphy (2012, ch. 16) [9].

Let $b_m(\mathbf{X}) : \mathbb{R}^p \mapsto \mathbb{R}$ denote the mth basis expansion of $\mathbf{X}$, for $m = 1, 2, ..., M$. The *linear basis expansion* is on the form

$$
\begin{aligned}
f(\mathbf{X}) &= \sum_{m=1}^{\mathcal{M}} \beta_m b_m(\mathbf{X}) \\
&= \sum_{m=1}^{\mathcal{M}} f_m(\mathbf{X}),
\end{aligned} \tag{14}
$$

which, once the basis functions b_m have been determined, is linear in terms of these $\mathcal{M}$ functions of $\mathbf{X}$ and the coefficient vector β can be estimated using least squares method.

Many models can be represented as a linear basis expansion by appropriate definitions of the basis functions. We are particularly interested in the models where the basis functions are parametric. In this case we can write $b_m(\mathbf{x}) = b(\mathbf{x}; \mathbf{v}_m)$, where $\mathbf{v}_m$ characterize the mth basis function. This means that each basis function is fit to data. The model is no longer linear in the model parameters $(\beta_{1:\mathcal{M}}, \{\mathbf{v_m}\}_1^{\mathcal{M}})$ and the corresponding RSS minimization problem (13) requires iterative methods or numerical optimization (Hastie et al., 2009, pp. 30; Murphy, 2012, pp. 544) [8, 9]. Typically, such methods produce a dictionary of basis functions, which is combined with a method for controlling model complexity. We shall see that dictionary methods with stagewise greedy selection methods such as CART (section 3.3.3) and Boosting (section 3.4) are of special interest in this thesis.

3.3.3 Regression Trees

Classification and Regression Trees (CART), introduced by Breiman et al. (1984) [19], is a popular type of tree-based method applicable to both classification and regression problems. We will restrict ourselves to CART in a regression setting, often called *regression trees*. We follow the content in Hastie et al. (2009, section 9.2.1-2) [8].

The basic idea of tree-based methods are to partition the domain of $\mathbf{X}$ into disjoint regions and then assess a certain response in each region. CART is restricted to recursive binary splits parallel to the coordinate axes, of which the response is the mean of Y corresponding to each region.

Consider the linear basis expansion (14). By letting $\mathcal{M} = J$, and correspondingly $\beta_j = \gamma_j$ and $b(\mathbf{X}; \mathbf{v}_j) = \mathbb{1}\{(X_1, X_2, ..., X_p) \in R_j\}$, we have defined the regression tree model

$$
f(\mathbf{X}) = \sum_{j=1}^{J} \gamma_j \mathbb{1}\{(X_1, X_2, ..., X_p) \in R_j\}, \tag{15}
$$

where γ_j is the response corresponding to region j, and $\mathbb{1}$ is the indicator function of the event that $\mathbf{X}$ is in region j. The tree parameters $\mathbf{v}_j$ encode the choice of split variables k, and split points s, on the path from the root node to the jth leaf.

3.3.3.1 Growing a Regression Tree

The process of partitioning the domain of $\mathbf{X}$ is referred to as growing the tree. CART implements a greedy algorithm which we will now present.

Consider a training set $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, and suppose that we have a partition into regions $R_1, R_2, ..., R_J$, as in (15). Then $f(\mathbf{X})$ is linear in J specified tree basis functions and we can minimize the RSS criteria (13) to obtain the optimal coefficients $\{\hat{\gamma}_j\}_1^J$. The solution turn out to equal the average of y_i corresponding to each region R_j, that is, $\hat{\gamma}_j = \text{ave}(y_i|\mathbf{x}_i \in R_j)$.

Estimating the split points and splitting variables, is done in a greedy fashion since solving the RSS criteria is generally infeasible. Starting with the whole training set $\mathcal{T}$, we consider a splitting variable k and a split point s, and define the half-planes

$$R_1(k,s) = \{\mathbf{X}|\mathbf{X}_k \leq s\} \quad \text{and} \quad R_2(k,s) = \{\mathbf{X}|\mathbf{X}_k > s\}. \tag{16}$$

The splitting variable and split point are those who solve

$$\min_{k,s}\left[\min_{\gamma_1}\sum_{\mathbf{x}_i\in R_1(k,s)}(y_i-\gamma_1)^2 + \min_{\gamma_2}\sum_{\mathbf{x}_i\in R_2(k,s)}(y_i-\gamma_2)^2\right], \tag{17}$$

where the inner minimization is solved by

$$\hat{\gamma}_1 = \text{ave}\{y_i|\mathbf{x}_i \in R_1(k,s)\} \quad \text{and} \quad \hat{\gamma}_2 = \text{ave}\{y_i|\mathbf{x}_i \in R_2(k,s)\}, \tag{18}$$

for every (k,s). Thus, we are left with solving

$$\min_{k,s}\left[\sum_{\mathbf{x}_i\in R_1(k,s)}(y_i-\hat{\gamma}_1)^2 + \sum_{\mathbf{x}_i\in R_2(k,s)}(y_i-\hat{\gamma}_2)^2\right]. \tag{19}$$

The best split pair (k,s) is obtained by scanning through all possible pairs from the input data. Having obtained such pair, they define the partitions R_1 and R_2, and this process is repeated on these regions until some stopping criteria is met.

Tree size J, that is, how large a tree should grow is a meta-parameter governing the model complexity. There exist several methods to determine the tree size. However, when trees are implemented as weak learners in a boosting algorithm, we shall later see that tree size becomes a meta-parameter in the boosting procedure itself which can be determined using cross-validation.

3.4 Boosting

Boosting (Shapire 1990; Freud & Shapire 1997) [20] is a greedy algorithm for fitting adaptive basis expansions on the form (14), where the basis functions $b(\mathbf{x};\mathbf{v}_m)$ are simple functions generated by a weak learner or base learner which is a predictor that performs only slightly better than chance (Hastie et al., 2009, section 10.2; Murphy 2012, sect. 16.4) [9, 8]. The boosting algorithm applies the

weak learner sequentially and the final prediction consists of a weighted majority vote from the $\mathcal{M}$ weak learners. Boosting was first presented in computational learning theory where it achieved great success. Friedman et al. (2000) [21] then presented a statistical view of Boosting, which explained its success as a procedure to fit additive models in a forward stagewise manner.

In general, boosting aims to solve

$$\{\hat{\beta}_m, \hat{\mathbf{v}}_m\}_1^{\mathcal{M}} = \arg \min_{\{\beta_m, \mathbf{v}_m\}_1^{\mathcal{M}}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{\mathcal{M}} \beta_m b(\mathbf{x}_i; \mathbf{v}_m)\right). \tag{20}$$

which, as mentioned in section 3.3.2, requires computationally intense numerical optimization techniques for squared error loss.

3.4.1 Forward Stagewise Additive Modeling

Forward Stagewise Additive Modelling (FSAM) (Hastie, 2009, sect. 10.2-3) [8] is a method that can be used to approximate (20), when it is possible to rapidly estimate the parameters of a single basis function

$$(\hat{\beta}, \hat{\mathbf{v}}) = \arg \min_{(\beta, \mathbf{v})} \sum_{i=1}^{N} L\left(y_i, \beta b(\mathbf{x}_i; \mathbf{v})\right). \tag{21}$$

This procedure is defined in Algorithm 1, where it is shown that each iteration or boost m add the optimal parametric basis function and the corresponding coefficient to the current expansion $\hat{f}^{(m-1)}$. Here optimal is in terms of minimizing the expected prediction error.

Input:
A data set $\mathcal{T}$
A loss function L
A base learner $b(\mathbf{x}; \mathbf{v})$
The number of iterations $\mathcal{M}$
1 Initialize $\hat{f}^{(0)}(\mathbf{x}) = \hat{f}_0(\mathbf{x}) = 0$;
2 **for** $m = 1$ *to* $\mathcal{M}$ **do**
3 $\quad (\hat{\beta}_m, \hat{\mathbf{v}}_m) = \arg\min_{\{\beta, \mathbf{v}\}} \sum_{i=1}^{N} L\left(y_i, \hat{f}^{(m-1)}(\mathbf{x_i}) + \beta b(\mathbf{x}_i; \mathbf{v})\right)$;
4 $\quad \hat{f}_m(\mathbf{x}) = \hat{\beta}_m b(\mathbf{x}, \hat{\mathbf{v}}_m)$;
5 $\quad \hat{f}^{(m)}(\mathbf{x}) = \hat{f}^{(m-1)}(\mathbf{x}) + \eta \hat{f}_m(\mathbf{x})$;
6 **end**
Output: $\hat{f}(\mathbf{x}) \equiv \hat{f}^{(\mathcal{M})}(\mathbf{x}) = \sum_{m=0}^{\mathcal{M}} \hat{f}_m(\mathbf{x})$

Algorithm 1: Forward Stagewise Additive Modeling (FSAM)

In case of least squares loss, the loss function in step 3 of Algorithm 1 can be expressed as

$$
\begin{aligned}
L\left(y_i, \hat{f}^{(m-1)}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \mathbf{v})\right) &= (y_i - \hat{f}^{(m-1)}(\mathbf{x}_i) - \beta b(\mathbf{x}_i; \mathbf{v}))^2 \\
&= (r_{im} - \beta b(\mathbf{x}_i; \mathbf{v}))^2,
\end{aligned}
$$

where $r_{im} = y_i - \hat{f}^{(m-1)}(\mathbf{x}_i)$ is the residuals of the current expansion $\hat{f}^{(m-1)}(\mathbf{x}_i)$. This means, that the mth boosting iteration chooses $\hat{f}_m(\mathbf{x}) = \hat{\beta}_m b(\mathbf{x}; \hat{\mathbf{v}}_m)$ that best fits the current residuals.

3.4.2 Newton Boosting

In this section we derive a boosting algorithm called Newton Boosting, which is a generic boosting algorithm that approximates the optimization problem in step 3 in Algorithm 1 for any twice differentiable loss function. This boosting algorithm is implemented by XGBoost.

3.4.2.1 Background

Boosting, as described above, was limited to certain loss functions for which a computationally feasible method for solving (21) existed. Friedman (2001) [10] then viewed boosting in terms of steepest descent in function space, and presented a generic boosting procedure called Gradient Boosting, applicable for any sub-differentiable loss function.

Newton boosting, which we will present in the next section, relies on the same underlying idea but instead of gradient descent implement Newton's Method in function space. Actually, Gradient Boosting is a special case of Newton Boosting.

We will not go into the details of numerical optimization in function space, or derive the Gradient Boosting algorithm. The interested reader is referred to above mentioned article by Friedman, in which he generalizes numerical optimization in parameter space to numerical optimization in function space.

3.4.2.2 Newton Boosting

Consider the optimization problem of choosing the next boost step $\hat{f}(\mathbf{x})$ in step 3 in Algorithm 1. We shall now see how Newton boosting chooses the mth boost step as the basis function $b(\mathbf{x}; \mathbf{v}_m)$ that produces $\{b(x_i; \mathbf{v}_m)\}_1^N$ most correlated to the negative gradient Hessian ratio in the N-dimensional data space at the current point $\hat{f}^{(m-1)}(\mathbf{x})$.

We will first define the general version of Newton Boosting, and then derive its sample counterpart. The content is mainly based on *Gradient and Newton Boosting for Classification and Regression* by Sigrist (2019) [4].

Let $Y \in \mathbb{R}$ and $\mathbf{X} \in \mathbb{R}^p$ be stochastic variables, let $\mathcal{S}$ be a span of base learners, and let $R(f) = E_{\mathbf{X},Y}(L(Y, f(\mathbf{X})))$, that is, the risk function or expected loss. Newton boosting chooses $\hat{f}_m$ as the minimizer of a second order Taylor approximation around the current point $\hat{f}^{(m-1)}$, consequently

$$
\begin{aligned}
\hat{f}_m &= \arg\min_{f\in\mathcal{S}} R(\hat{f}^{(m-1)} + f) \\
&= \arg\min_{f\in\mathcal{S}} R(\hat{f}^{(m-1)}) + \frac{\partial R(\hat{f}^{(m-1)})}{\partial \hat{f}^{(m-1)}} + \frac{1}{2}\frac{\partial^2 R(\hat{f}^{(m-1)})}{\partial (\hat{f}^{(m-1)})^2}.
\end{aligned} \tag{22}
$$

Assuming P-almost all existence and integrability of the second derivative of the loss function $L(Y, f)$ with respect to f, we can interchange differentiation and integration such that (22) can be written as

$$
\begin{aligned}
\hat{f}_m &= \arg\min_{f\in\mathcal{S}} R(\hat{f}^{(m-1)}) + \frac{\partial R(\hat{f}^{(m-1)})}{\partial f} + \frac{1}{2}\frac{\partial^2 R(\hat{f}^{(m-1)})}{\partial f^2} \\
&= \arg\min_{f\in\mathcal{S}} E_{Y,\mathbf{X}}\left[g_m(Y,\mathbf{X})f(\mathbf{X}) + \frac{1}{2}h_m(Y,\mathbf{X})f^2(\mathbf{X})\right] \\
&= \arg\min_{f\in\mathcal{S}} E_{Y,\mathbf{X}}\left[h_m(Y,\mathbf{X})\left(-\frac{g_m(Y,\mathbf{X})}{h_m(Y,\mathbf{X})} - f(\mathbf{X})\right)^2\right],
\end{aligned} \tag{23}
$$

where we have removed the constant $R(\hat{f}^{(m-1)})$ which does not affect the minimization, and $g_m(\mathbf{X}, Y)$ and $h_m(\mathbf{X}, Y)$ are the gradient and Hessian of the loss function $L(Y, f)$ with respect to the current estimate $\hat{f}^{(m-1)}$, that is,

$$
\begin{aligned}
g_m(\mathbf{X}, Y) &= \left[\frac{\partial L(\mathbf{X}, f)}{\partial f}\right]_{f=\hat{f}^{(m-1)}(\mathbf{X})}, \\
h_m(\mathbf{X}, Y) &= \left[\frac{\partial^2 L(\mathbf{X}, f)}{\partial f^2}\right]_{f=\hat{f}^{(m-1)}(\mathbf{X})}.
\end{aligned}
$$

The last row in equation (23) shows that $\hat{f}_m$ is the weighted least squares approximation to $-\frac{g_m(Y,\mathbf{X})}{h_m(Y,\mathbf{X})}$, where the weights are given by the Hessian $h_m(Y, \mathbf{X})$. The factor $-\frac{g_m(Y,\mathbf{X})}{h_m(Y,\mathbf{X})}$ is commonly referred to as the Newton step.

If the following expression is well defined for P-almost all $\mathbf{X}$, we can calculate (23) with respect to the conditional risk

$$
\hat{f}_m(\mathbf{X}) = \arg\min_{f\in\mathcal{S}} E_{Y|\mathbf{X}}\left[h_m(Y,\mathbf{X})\left(-\frac{g_m(\mathbf{X},Y)}{h_m(\mathbf{X},Y)} - f(\mathbf{X})\right)^2\right].
$$

If we now consider a training set $\mathcal{T}$ of N independent input-output pairs from the distribution of $(\mathbf{X}, Y)$, and approximate the risk $R(f)$ with the empirical risk for squared error loss $R^e(f) = \sum_{i=1}^N L(y_i, f(\mathbf{x}_i)) = \sum_{i=1}^N (y_i - f(\mathbf{x}_i))^2$, then the Newton update in equation (23) is given by

$$
\begin{aligned}
\hat{f}_m &= \arg\min_{f \in \mathcal{S}} \sum_{i=1}^{N} g_{im} f(\mathbf{x}_i) + \frac{1}{2} h_{im} f(\mathbf{x}_i)^2 \\
&= \arg\min_{f \in \mathcal{S}} \sum_{i=1}^{N} h_{im} \left(-\frac{g_{im}}{h_{im}} - f(\mathbf{x}_i) \right)^2 ,
\end{aligned} \tag{24}
$$

where g_{im} and h_{im} are the empirical gradient and Hessian for observation i, that is,

$$
\begin{aligned}
g_{im} &= \left[\frac{\partial}{\partial f} L(y_i, f) \right]_{f = \hat{f}^{(m-1)}(\mathbf{x}_i)} , \\
h_{im} &= \left[\frac{\partial^2}{\partial f^2} L(y_i, f) \right]_{f = \hat{f}^{(m-1)}(\mathbf{x}_i)} .
\end{aligned}
$$

The Newton Boosting algorithm is summarized in Algorithm 2.

Input: Data set $\mathcal{T}$
A loss function L
A base learner $b(\mathbf{x}; \mathbf{v})$
The number of iterations $\mathcal{M}$
The learning rate η
Initialize $\hat{f}^{(0)}(\mathbf{x}) = \hat{f}_0(\mathbf{x}) = \hat{\beta} = \arg\min_\beta \sum_{i=1}^{n} L(y_i, \beta)$;
for $m = 1$ *to* $\mathcal{M}$ **do**
  $\hat{g}_m(\mathbf{x}_i) = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f(\mathbf{x}_i) = \hat{f}^{(m-1)}(\mathbf{x}_i)}$;
  $\hat{h}_m(\mathbf{x}_i) = \left[\frac{\partial^2 L(y_i, f(\mathbf{x}_i))}{\partial f^2(\mathbf{x}_i)} \right]_{f(\mathbf{x}_i) = \hat{f}^{(m-1)}(\mathbf{x}_i)}$;
  $\hat{b}_m = \arg\min_{b \in \mathcal{B}} \sum_{i=1}^{n} \frac{1}{2} \hat{h}_m(\mathbf{x}_i) \left[\left(-\frac{\hat{g}_m(\mathbf{x}_i)}{\hat{h}_m(\mathbf{x}_i)} \right) - b(\mathbf{x}_i) \right]^2$;
  $\hat{f}_m = \hat{b}_m(\mathbf{x})$;
  $\hat{f}^{(m)}(\mathbf{x}) = \hat{f}^{(m-1)}(\mathbf{x}) + \eta \hat{f}_m(\mathbf{x})$;
end
Output: $\hat{f}(\mathbf{x}) \equiv \hat{f}^{(\mathcal{M})}(\mathbf{x}) = \sum_{m=0}^{\mathcal{M}} \hat{f}_m(\mathbf{x})$

Algorithm 2: Newton Boosting

3.4.3 Boosting Parameters

Fitting data too closely can be counterproductive and lead to decreased generalization performance, as derived in section 3.2. Regularization methods attempt to prevent over-fitting by constraining the fitting procedure. Boosting has two primary meta-parameters.

The Number of Iterations $\mathcal{M}$

A natural constraint for additive expansions on the form (14) is the number of

basis expansions $\mathcal{M}$, which is the main meta-parameter of the boosting procedure in Algorithm 1. Controlling the number of terms in the expansion places an implicit prior belief that "sparse" approximations involving fewer terms are likely to provide better prediction (Friedman, 2001) [10].

The Learning Rate η

In his 2001 article (Friedman, 2001) [10], Friedman also suggested using a shrinkage parameter η in the boosting procedure, such that line 5 in Algorithm 1 is replaced by

$$\hat{f}^{(m)}(\mathbf{x}) = \hat{f}^{(m-1)}(\mathbf{x}) + \eta \hat{f}_m(\mathbf{x}), \quad 0 < \eta \leq 1.$$

Friedman argues that $\mathcal{M}$ and η are not independent, thus they should be fitted simultaneously. In general, empirical studies show that $\eta < 0.1$ and large values of $\mathcal{M}$ increase model performance. Unfortunately, this "slow learning" increases computational load. It is thus recommended to choose a combination of these variables such that the computations can be performed in a reasonable amount of time. Newton Boosting implements shrinkage, which can be seen at line 7 in Algorithm 2.

3.5 XGBoost

In this section we will apply the theory in previous sections to derive eXtreme Gradient Boosting (XGBoost) by Chen & Guestrin (2016) [3]. We start by deriving Newton Boosting (Algorithm 2) in the case of using regression trees as base learners and squared error loss. We then move on to the regularized learning objective of XGBoost, which includes l_1 and l_2 penalty terms on objective and define the algorithm used in this thesis.

3.5.1 Boosting Trees

When choosing regression trees on the form (15) as basis functions in the parametric version of the basis expansion (14), we get

$$\begin{aligned} f(\mathbf{X}) &= f^{(\mathcal{M})}(\mathbf{X}) \\ &= \sum_{m=1}^{\mathcal{M}} \beta_m \sum_{j=1}^{J} \gamma_{jm} \mathbb{1}(\mathbf{X} \in R_{jm}) \\ &= \sum_{m=1}^{\mathcal{M}} \sum_{j=1}^{J} w_{jm} \mathbb{1}(\mathbf{X} \in R_{jm}) \\ &= \sum_{m=1}^{\mathcal{M}} T(\mathbf{X}; \Theta_m) \\ &= \sum_{m=1}^{\mathcal{M}} f_m(\mathbf{X}), \end{aligned} \tag{25}$$

where $w_{jm} = \beta_m \gamma_{jm}$ is the response or weight corresponding to each region R_{jm}, and $\Theta_m = \{R_{jm}, w_{jm}\}_1^{J_m}$ is the parameter set of the mth tree $T(\mathbf{X}; \Theta_m)$. We can see, at the second row in (25), that boosted trees is itself an ensemble of additive trees. Remembering from section 3.3.3 that the response for a given tree structure equals the coefficient w_{jm} for which the input $\mathbf{X} \in R_{jm}$, we realise that the response for a specified function $f^{(\mathcal{M})}(\mathbf{X})$ is a sum of the corresponding $\mathcal{M}$ region weights.

3.5.2 Newton Tree Boosting

In this section we will derive a general XGBoost algorithm which fit tree ensembles on the form (25), and later we shall see how it is modified by introducing a penalized learning objective.

Given a training set $\mathcal{T}$ of length n and a twice differentiable, convex loss function L, Newton Boosting solves

$$\{\hat{\mathbf{\Theta}}_m\}_1^{\mathcal{M}} = \arg\min_{\{\mathbf{\Theta}_m\}_1^{\mathcal{M}}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{\mathcal{M}} T(\mathbf{x}_i; \mathbf{\Theta}_m)\right) \tag{26}$$

in a forward stagewise manner. Considering the stagewise minimization problem on the form of (24) where we assume a given tree structures gives us

$$\begin{aligned}
\hat{\mathbf{\Theta}}_m &= \arg\min_{\mathbf{\Theta}} \sum_{i=1}^{N} \left[g_{im} T(\mathbf{x}_i; \mathbf{\Theta}) + \frac{1}{2} h_{im} T(\mathbf{x}_i; \mathbf{\Theta})^2 \right] \\
&= \arg\min_{\mathbf{\Theta}} \sum_{i=1}^{N} \left[g_{im} \sum_{j=1}^{J_m} w_{jm} \mathbb{1}(\mathbf{x}_i \in R_{jm}) + \frac{1}{2} h_{im} \sum_{j=1}^{J_m} w_{jm}^2 \mathbb{1}(\mathbf{x}_i \in R_{jm})^2 \right] \\
&= \arg\min_{\mathbf{\Theta}} \sum_{j=1}^{J} \left[\left(\sum_{\mathbf{x}_i \in R_{jm}} g_{im} \right) w_{jm} + \frac{1}{2} \left(\sum_{\mathbf{x}_i \in R_{jm}} h_{im} \right) w_{jm}^2 \right] \\
&= \arg\min_{\mathbf{\Theta}} \sum_{j=1}^{J} \left[G_{jm} w_{jm} + \frac{1}{2} H_{jm} w_{jm}^2 \right],
\end{aligned} \tag{27}$$

where G_{jm} and H_{jm} denote the gradient and the Hessian sum respectively, and in the third equality we used additivity of the disjoint regions. Finding the solutions to

$$\frac{\partial}{\partial w_{jm}} \sum_{j=1}^{J} \left[G_{jm} w_{jm} + \frac{1}{2} H_{jm} w_{jm}^2 \right] = \sum_{j=1}^{J} \left[G_{jm} + H_{jm} w_{jm} \right] = 0$$

gives us the optimal leaf weights, which are given by

$$\hat{w}_{jm} = -\frac{G_{jm}}{H_{jm}}.$$

Inserting these weights into the last expression in equation (27) gives us a score on the optimized tree with a given structure

$$Score(\hat{\mathbf{\Theta}}_m) = \sum_{j=1}^{J} -\frac{G_{jm}^2}{H_{jm}} + \frac{1}{2}\frac{G_{jm}^2}{H_{jm}} = -\frac{1}{2}\sum_{j=1}^{J}\frac{G_{jm}}{H_{jm}}.$$

The score is used to find an optimal tree structure by evaluating possible node splits. XGBoost uses a greedy top-down split finding algorithm, similar to CART which we described in section 3.3.3. The algorithm consider the possible binary splits of a region R into R_L and R_R, and define the *gain* in terms of reduced empirical risk of a specific split. The gain is calculated as

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right], \tag{28}$$

where the first two terms correspond to the score on the left and right leaf respectively, and the last term corresponds to the original score. The split algorithm perform the split that yield the largest gain, given that $Gain > 0$.

This general XGBoost algorithm is summarized in Algorithm 3.

Input: Data set $\mathcal{T}$
A loss function L
The number of iterations $\mathcal{M}$
The learning rate η
Tree size J
Initialize $\hat{f}^{(0)}(\mathbf{x}) = \hat{f}_0(\mathbf{x}) = \hat{\beta} = \arg\min_\beta \sum_{i=1}^n L(y_i, \beta)$;
for $m = 1$ *to* $\mathcal{M}$ **do**
  $\hat{g}_{im}(\mathbf{x}_i) = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f(\mathbf{x}_i)=\hat{f}^{(m-1)}(\mathbf{x}_i)}$;
  $\hat{h}_{im}(\mathbf{x}_i) = \left[\frac{\partial^2 L(y_i, f(\mathbf{x}_i))}{\partial f^2(\mathbf{x}_i)}\right]_{f(\mathbf{x}_i)=\hat{f}^{(m-1)}(\mathbf{x}_i)}$;
  Determine the structure $\{\hat{R}_{jm}\}_1^J$ by selecting splits which maximize $Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right]$;
  Determine the leaf weights $\{\hat{w}_{jm}\}_1^J$ for the learned structure by $\hat{w}_{jm} = -\frac{G_{jm}}{H_{jm}},\ j = 1, 2, ..., J$;
  $\hat{f}^{(m)(\mathbf{x})} = \sum_{j=1}^J \hat{w}_{jm}\mathbb{1}(\mathbf{x}_i \in \hat{R}_{jm}) = T(\mathbf{x}; \hat{\mathbf{\Theta}}_m)$;
  $\hat{f}^{(m)}(\mathbf{x}) = f^{(m-1)} + \eta\hat{f}_m(\mathbf{x})$;
end
Output: $\hat{f}(\mathbf{x}) \equiv \hat{f}^{(\mathcal{M})}(\mathbf{x}) = \sum_{m=0}^{\mathcal{M}} \hat{f}_m(\mathbf{x})$

Algorithm 3: XGBoost

3.5.3 Penalized Learning Objective

Chen & Guestrin (2016) [3] introduce a penalized learning objective to prevent over-fitting, which we will now present.

The penalized counterpart to (26) is on the form

$$\{\hat{\mathbf{\Theta}}_m\}_1^{\mathcal{M}} = \arg\min_{\{\mathbf{\Theta}_m\}_1^{\mathcal{M}}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{\mathcal{M}} T(\mathbf{x}_i; \mathbf{\Theta}_m)\right) + \sum_{m=1}^{\mathcal{M}} \Omega(\mathbf{\Theta}_m)$$

where $\Omega(\mathbf{\Theta}_m)) = \Gamma J + \alpha \sum_{j=1}^{J} |w_{jm}| + \frac{1}{2}\lambda \sum_{j=1}^{J} w_{jm}^2$ are the penalization terms for the number of terminal nodes J, l_1 (lasso)[1] and l_2 (ridge) regularization respectively. This result in stagewise minimization, corresponding to (27), on the form

$$\begin{aligned}
\hat{\mathbf{\Theta}}_m &= \arg\min_{\mathbf{\Theta}} \sum_{i=1}^{N} \left[g_{im} T(\mathbf{x}_i; \mathbf{\Theta}) + \frac{1}{2} h_{im} T(\mathbf{x}_i; \mathbf{\Theta})^2 \right] + \Gamma J + \alpha \sum_{j=1}^{J} |w_{jm}| + \frac{1}{2}\lambda \sum_{j=1}^{J} w_{jm}^2 \\
&= \arg\min_{\mathbf{\Theta}} \sum_{j=1}^{J} \left[\left(\sum_{\mathbf{x}_i \in R_{jm}} g_{im} \right) w_{jm} + \frac{1}{2} \left(\sum_{\mathbf{x}_i \in R_{jm}} h_{im} + \lambda \right) w_{jm}^2 + \alpha |w_{jm}| \right] + \Gamma J \\
&= \arg\min_{\mathbf{\Theta}} \sum_{j=1}^{J} \left[G_{jm} w_{jm} + \frac{1}{2} \left(H_{jm} + \lambda \right) w_{jm}^2 + \alpha |w_{jm}| \right] + \Gamma J.
\end{aligned} \tag{29}$$

We continue by deriving an expression for the optimal leaf weights $\{\hat{w}_{jm}\}_1^J$, which is the solution to

$$\begin{aligned}
&\frac{\partial}{\partial w_{jm}} \sum_{j=1}^{J} \left[G_{jm} w_{jm} + \frac{1}{2} \left(H_{jm} + \lambda \right) w_{jm}^2 + \alpha |w_{jm}| \right] + \Gamma J = 0 \\
\Rightarrow &\frac{\partial}{\partial w_{jm}} \sum_{j=1}^{J} \left[G_{jm} + \left(H_{jm} + \lambda \right) w_{jm} + sign(\alpha) \right] = 0
\end{aligned}.$$

For a given tree structure at iteration m, assuming $w_{jm} \leq 0$, we get

$$\begin{aligned}
&\frac{\partial}{\partial w_{jm}} G_{jm} + \left(H_{jm} + \lambda \right) w_{jm} - \alpha = 0 \\
\Rightarrow &\hat{w}_{jm} = -\frac{G_{jm} - \alpha}{H_{jm} + \lambda}
\end{aligned}$$

,

[1] Chen & Guestrin (2016) does not include the l_1 regularization term. However, this term is included in the `xgboost`-package in which XGBoost is implemented in R. We choose to include this term here.

where the denominator is strictly positive and thus $G_{jm} - \alpha \geq 0$. Solving under the assumption that $w_{jm} > 0$ is done analogously, and we get the following solution

$$\hat{w}_{jm} = \begin{cases} -\frac{G_{jm}+\alpha}{H_{jm}+\lambda}, & \text{if } G_{jm} < -\alpha \\ -\frac{G_{jm}-\alpha}{H_{jm}+\lambda}, & \text{if } G_{jm} > \alpha \\ 0, & \text{else} \end{cases}.$$

Inserting in (29) results in the following score expression

$$Score(\hat{\mathbf{\Theta}}_m) = -\frac{1}{2}\sum_{j=1}^{J} \frac{\mathcal{A}_\alpha^2}{H_{jm}+\lambda} + \Gamma J.$$

where

$$\mathcal{A}_\alpha(G_{jm}) = \begin{cases} G_{jm}+\alpha, & \text{if } G_{jm} < -\alpha \\ G_{jm}-\alpha, & \text{if } G_{jm} > \alpha \\ 0, & \text{else.} \end{cases}$$

Finally, we can express the gain as

$$Gain = \frac{1}{2}\left[\frac{\mathcal{A}_\alpha(G_{jmL})G_{jmL}^2}{H_{jmL}+\lambda} + \frac{\mathcal{A}_\alpha(G_{jmR})G_{jmR}^2}{H_{jmR}+\lambda} - \frac{\mathcal{A}_\alpha(G_{jm})G_{jm}^2}{H_{jm}+\lambda}\right] - \Gamma. \quad (30)$$

3.5.4 XGBoost Parameters

In section 3.4.3 we defined the boosting meta-parameters $\mathcal{M}$ and η. In this section, we shall define the remaining main parameters for the XGBoost method.

3.5.4.1 Tree Parameters

Tree Size J

This parameter relates to the order of interactions among the predictor variables, and the optimal value is such that it reflects the number of effective interaction order of the target function $f(\mathbf{X})$ in (8), and is often found using cross-validation (Friedman, 2001, section 7) [10]. Empirical studies (Friedman, 2001; Probst et al, 2018) [10, 22]) suggest collectively the use of trees of size 6-15.

Minimum Child Weight ω

The minimum sum of instance weight (Hessian) needed in a child, or leaf. If the tree partition step results in a leaf node with the sum of instance weight less than minimum child weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances or observations needed to be in each node. The larger, the more conservative the algorithm will be (Chen et al., 2020) [23].

Gamma Γ

Used to reduce *Gain* in possible splits, as can be seen in equations (28) and (30). This parameter is not considered in this thesis.

3.5.4.2 Complexity Penalization

Chen & Guestrin (2016) [3] introduce a penalized learning objective, which we derived in section 3.5.3. This type of regularization method relies on a prior belief that the target exhibit a type of smooth behaviour (Hastie et al., 2009, pp. 34) [8].

l_2 Regularization λ

The λ parameter controls the strength of the l_2 (Ridge) penalization term, which shrinks the leaf weights. This becomes clear by presenting the *Gain* and final leaf weights $\hat{w}_{jm}$, analogously to in Section 3.5.3, to a model only including l_2 regularization, which become

$$Gain = \frac{1}{2}\left[\frac{G_{jmL}^2}{H_{jmL}+\lambda} + \frac{G_{jmR}^2}{H_{jmR}+\lambda} - \frac{G_{jm}^2}{H_{jm}+\lambda}\right] - \Gamma,$$

and

$$\hat{w}_{jm} = -\frac{G_{jm}}{H_{jm}+\lambda}.$$

Since λ have an effect on the *Gain*, it may also impact the tree structure.

l_1 Regularization α

The α parameter controls the strength of the l_1 (LASSO) penalization term, which is similar to l_2 regularization. The *Gain* and final leaf weights $\hat{w}_{jm}$, analogously to l_2 regularization above, now become

$$Gain = \frac{1}{2}\left[\frac{\mathcal{A}_\alpha(G_{jmL})G_{jmL}^2}{H_{jmL}} + \frac{\mathcal{A}_\alpha(G_{jmR})G_{jmR}^2}{H_{jmR}} - \frac{\mathcal{A}_\alpha(G_{jm})G_{jm}^2}{H_{jm}+\lambda}\right],$$

and

$$\hat{w}_{jm} = -\frac{\mathcal{A}_\alpha(G_{jm})}{H_{jm}}.$$

Thus, also λ have an effect on the *Gain*, it may also impact the tree structure. Unlike l_2 regularization, l_1 regularization can shrink leaf weights to zero.

3.5.4.3 Randomization Parameters

Row Subsampling δ_r

Friedman (2002) [11] suggest that introducing row subsampling in each iteration would increase generalization performance and computational efficiency.

This can be interpreted as a type of regularization. Row subsampling means that only a proportion δ_r of the N training observations are used in order to determine the next addition $\hat{f}^{(m)}$.

Column Subsample By Tree δ_c
In addition to row subsampling, XGBoost supports column, or feature, subsampling. This technique should prevent over-fitting and increases speed by reducing computations. The parameter δ_c is the proportion of features kept.

3.5.5 Relative Importance

Relative importance or feature importance allows for assessing how the input features contribute to the response, in a relative perspective. The R package `xgboost` which we use to implement XGBoost in section 4 offers this functionality which is described in Hastie et al. (2009, section 10.13.1).

For a single decision tree T, Brieman et al. (1984) [19] proposed

$$\mathcal{I}_l^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 I(v(t) = l) \tag{31}$$

as a relevance measure for each predictor X_l. Here $J-1$ equal the number of internal nodes, v_t is the splitting variable associated with node t, and $\hat{i}_t^2$ is the corresponding empirical risk improvement in squared error as a result of the split. At each node t, one of the input variables $X_{v(t)}$ is used to partition the region associated with that node into two subregions and within each of those regions a constant is fit to the response values. The particular input variable that gives the maximal estimated improvement $\hat{i}_t^2$ is selected.

The importance measure (31) can be generalized to additive tree expansions by

$$\mathcal{I}_l^2(T) = \frac{1}{\mathcal{M}} \sum_{t=1}^{J-1} \mathcal{I}_l^2(T_m),$$

which is the average over the $\mathcal{M}$ trees. The averaging stabilizes the measure which makes it more reliable than (31).

3.6 Supervised Learning Setting to Model Time Dependencies

In this section we will present a general machine learning approach to one-step forecasting of univariate time series. The approach is mainly based on *Machine Learning Strategies for Time Series Forecasting* by Bontempi et al. (2013) [1]. Bontempi also gives a general introduction to the supervised learning setting to model time dependencies and cite several related papers on this topic. Many of the early papers consider artificial neural network models (ANNs). Ahmed et al. (2010) [2] compared the forecasting capability of eight common supervised learning models on 1045 time series, where ANNs proved successful.

Consider $\{z_t\}_{t=1}^{N}$ to be a sample from a time series z_t. In a supervised learning forecasting setting, we can represent a training set of input-output pairs by an $(N-p-1)\times p$ input matrix $\mathbf{X}$ and an $(N-p-1)\times 1$ output vector Y

$$Y = \begin{bmatrix} z_N \\ z_{N-1} \\ \vdots \\ z_{p+1} \end{bmatrix}, \mathbf{X} = \begin{bmatrix} z_{N-1} & z_{N-2} & \cdots & z_{N-p-1} \\ z_{N-2} & z_{N-3} & \cdots & z_{N-p-2} \\ \vdots & \vdots & \vdots & \vdots \\ z_p & z_{p-1} & \cdots & z_1 \end{bmatrix},$$

where p is the number of previous values. This type of rolling analysis of time series has traditionally been used to assess the stability of a model over time (Zivot, 2006, ch. 9) [24].

3.6.1 Cross-Validation for Dependent Data

One of the most common procedures for model evaluation in supervised regression is k-fold cross-validation (CV) which was derived in section 3.2.3. Using CV on time series in forecasting problems is not straight-forward, due to inherent serial correlation which contradicts assumptions about independent observations. Furthermore, the random partitioning process of CV entails using future values to predict past values which is not in line with the forecasting purpose. Bergmeir et al.[2] (2017) [5] study the implementation of three different types of model evaluation methods: CV, *Out-Of-Sample evaluation* (OOS), and *non-dependent Cross-Validation*. We will not derive the last two methods, but rather focus the remaining part of this subsection on some of Bergmeir's conclusions on applying CV in this type of problem.

From a purely theoretical perspective, CV on purely autoregressive models is valid if the residuals of the model is uncorrelated. Furthermore, an empirical study of autoregressive models, on Monte Carlo simulated data, show that CV, in its original design, outperforms OOS evaluation, which according to Bergmeir is the by practitioners assumed go-to method in forecasting model evaluation. Finally, for a real-world dataset CV can control over-fitting, and only if the model is under-fitting data or have heavily correlated errors should the method be avoided.

Bergmeir states that machine learning models that perform predictions on the form derived in the previous section, are considered such autoregressive models for which CV in general is the preferred evaluation method.

The interested reader is also referred to Bergmeir (2012) [25] which is a precursor to the above paper. Furthermore, Hart (1994) [26] present the *Time Series Cross-Validation* method (TSCV), which in short can be described as "evaluation on a rolling forecasting origin". Its design is close to the general prediction objective, allows for multi-step forecast evaluation and basis for estimating forecast uncertainty. The drawback is the large number of models to

[2]Co-author Rob J Hyndman is the creator of the `forecast` package which is used in this thesis to implement the ARMA modeling.

be fitted during the cross-validation process, which makes it an unlikely performance estimator in this thesis (see section 4.4.2).

4 Case Study

4.1 Data

For model comparison, we use a dataset consisting of daily closing prices of OMX Stockholm Large Cap Price Index (OMXSLCPI)[3] during the period 2015-01-04 to 2019-12-30. The index is market capitalization-weighted composition of all Large Cap companies listed on Nasdaq Stockholm. The dataset is considered to be of high quality since it has been obtained from a reliable first-hand source, it contains no missing values and visual inspection revealed no obvious inaccurate observations.

We follow common practice in financial time series analysis and use log returns rather than simple returns as input to the models (Tsay, 2010, pp. 83) [6]. However, experimental comparison of distributional properties on the specific dataset revealed only small differences which suggest that log transformation had little stabilizing effect, and that use of simple returns would lead to similar conclusions for modeling and performance comparison.

Before choosing to proceed with analysing log returns, we also considered using other types of Box-Cox stabilizing transformations on the original price series, and analysed the extent of seasonal patterns. Taking seasonal patterns into account is considered important when performing time series forecasting (Tsay, 2010, section 2.8) [6]. In general, there are two approaches to seasonal modeling. One is the use of seasonal differencing, and the other one is to use seasonal adjustment. We have excluded seasonal differencing, since trees cannot capture additive structures (Hastie et al., 2009, pp. 313) [8] indicating that the same applies for XGBoost. Using seasonal differencing in only the ARMA model, would bring uneven conditions when comparing model performance.

It is possible that other transformations and seasonally adjusting data, used either individually or combined, would have some positive effects on the predictive capabilities of the models. On the other hand, such pre-processing of data may have negative effect in terms of interpreting distributional properties of data and the final predicted result. These are the main reasons to why we chose to proceed with analysing log returns.

The log price and log return series are presented in Figure 2. We notice relatively large fluctuations of log returns during the beginning of the dataset. There is one significant outlier coinciding with the United Kingdom European Union membership referendum on the 24th of June 2016. Although this should reasonably be considered a single extreme event, this observation is retained.

[3] https://indexes.nasdaqomx.com/Index/Overview/OMXSLCPI

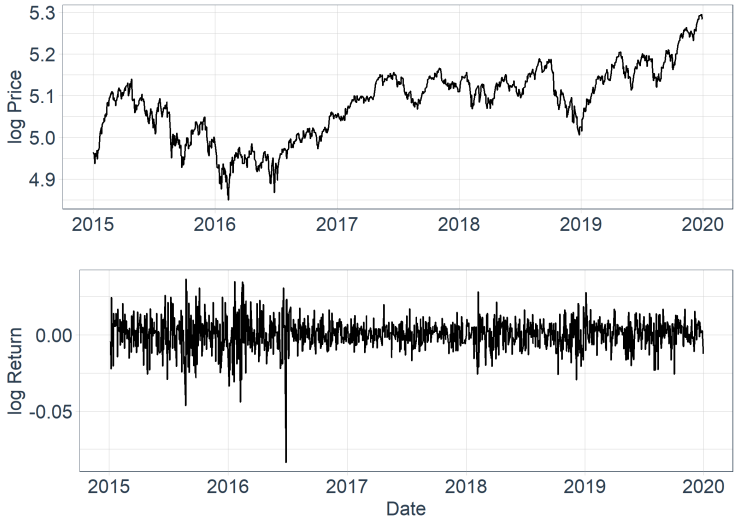

Figure 2: Daily OMX Stockholm Large Cap PI log prices and log returns, during the period 2015-01-04 to 2019-12-30.

4.1.1 Distributional properties

Distributional properties of the log returns are described in Table 1 and illustrated in Figure 3. We note that the mean return is close to zero. Conducting a one sample t-test shows that the hypothesis of zero expected return cannot be rejected at 95 % confidence level. Furthermore, according to the corresponding one sample t-tests (Tsay, 2010, sect. 1.2.1) [6], we can conclude with 95 % confidence that skewness and positive excess kurtosis of the log returns are significantly different from those of a normal distribution. The positive excess kurtosis property means log returns are heavy tailed, that is, they contain more extreme observations compared to a normal distribution. The heavy tail property is also indicated by the fluctuations in the Q-Q plot in Figure 3. Thus, the normal assumption of the log returns is questionable.

4.1.2 Stationarity

We have reason to believe that the log price series contain a unit root and is non-stationary, while the differenced log price series, that is, the log return series r_t, seems to fluctuate around a constant level which would suggest this is a weak stationary series, although the constant variation is questionable.

Conducting Augmented Dickey-Fuller unit-root test results in p-values of 0.41 and 0.01, for the log price- and log return- series respectively. Thus H_0: *existence of unit root* cannot be rejected for the log price series but can be rejected for the log return series. Henceforth we will regard the log return series to be weak stationary.

	$\mathbf{r_t}$	$\mathbf{R_t}$
Minimum	-0.083659	-0.080255
Maximum	0.036474	0.037147
Mean	0.000254	0.000303
Median	0.000665	0.000665
Stdev	0.009917	0.009890
Skewness	-0.648070	-0.550551
Kurtosis	5.172612	4.596897

Table 1: Descriptive statistics of daily OMX Stockholm Large Cap PI log return series r_t and simple return series R_t, during the period 2015-01-04 to 2019-12-30.

4.1.3 Dynamic Dependence

With our prediction purpose, it is essential to examine the dynamic dependence of the log return series. Visual assessment of the sample ACF and PACF in Figure 4 show that the serial dependence of daily log returns are weak. Most ACF:s and PACF:s are within two standard deviations, however presence of significant serial correlations are indicated at lags $4, 10$ and 24. The Ljung-Box test (Tsay, 2010, pp. 32) [6], using $m = \log(N - 1) = 7$ lags, gives a p-value of 0.008 which suggests existence of significant serial correlation, and indicates that the log return series can be used in prediction purpose.

4.2 Software

Code for data wrangling, analysis and modeling is entirely written in the language R using the RStudio IDE[4]. In addition to basic packages, some additional packages were used. The modeling of ARMA was implemented through the functions `Arima` and `auto.arima` from the `forecast` package developed by Hyndman et al. (2008, 2020) [27, 28]. The modeling of XGBoost was implemented by `xgb.train` and `xgb.cv` from the `xgboost` package developed by Chen & Guestrin (2016) [3]. Various functions from the `tidyverse` package, by Wickham et al. (2019) [29], were used for data wrangling and analysis, and in conjunction with above packages.

4.3 Outline of Model Selection and Model Comparison

In this section we give a background and overview of the approach taken for model selection, performance assessment and comparison of the XGBoost and ARMA models in the succeeding sections 4.4 and 4.5.

Since little is written about implementing XGBoost in the forecasting approach described in section 3.6, we mostly rely on traditional financial time series analysis and forecasting theory in Tsay (2010) [6], combined with theory

[4]Integrated Development Environment.

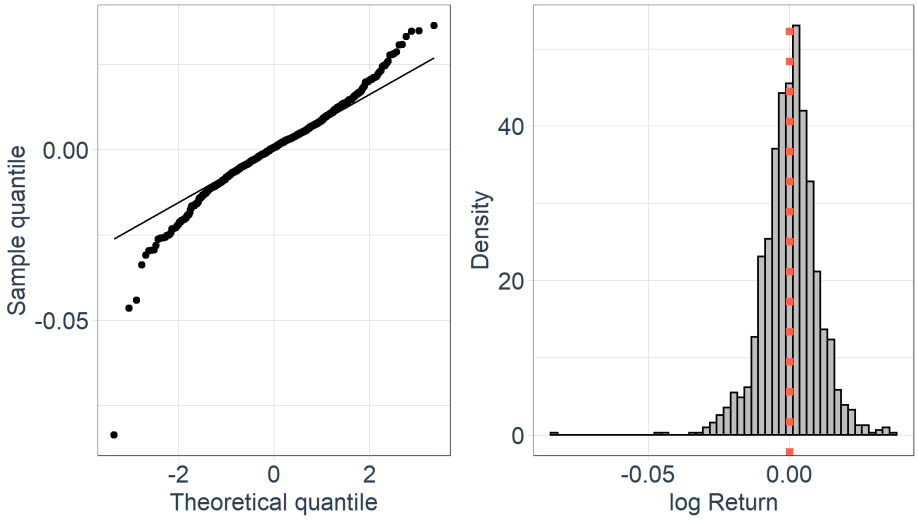

Figure 3: Q-Q plot and density plot of daily OMX Stockholm Large Cap PI log return series, during the period 2015-01-04 to 2019-12-30.

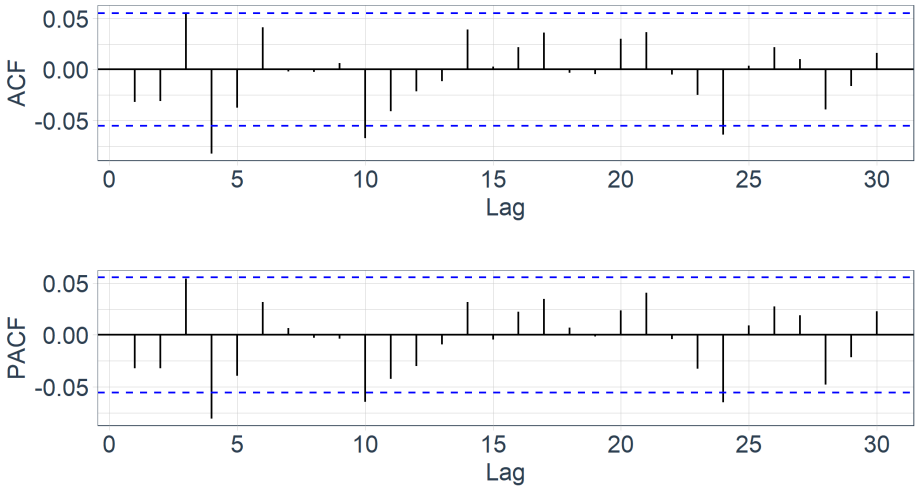

Figure 4: Q-Q plot and density plot of daily OMX Stockholm Large Cap PI log return series, during the period 2015-01-04 to 2019-12-30.

on statistical learning in Hastie et al. (2009) [8] and implement a rather basic approach. The idea is to provide base case, which opens up for improvement by implementation of alternative data pre-processing and model selection techniques that may improve the performance of XGBoost in this type of problem. A similar approach is taken in *An Empirical Comparison of Machine Learning Models for Time Series Forecasting*, by Ahmed et al. (2010, section 2) [2]. Some alternative suggestions are discussed in section 5.

4.3.1 Model Selection and Comparison

We partition the data sample into training- and test sets. Since the aim is forecasting future values, at this step, we do not randomly partition data. Instead we define the most recent year of data as our test set and the first four years of data as our training set. We then consider the training set to be historical values of which we will base our model selection. The test set is considered future unseen values.

The model selection for ARMA and XGBoost are conducted in section 4.4.1 and 4.4.2 respectively. We use different methods for model selection. The ARMA model selection procedure implements an information criteria combined with training set residual diagnostics, while the XGBoost selection procedure is based on k-fold cross-calidation. The reason to this is that these are common approaches for each method (Ahmed et al., 2010, section 4) [2].

Having selected the best models, we estimate their predictive performance by conducting one-step ahead forecasts on the test set. No re-estimation of model parameters is performed between steps. As performance measure we choose *root mean squared error* (RMSE), which is the square root of the *mean squared error* (MSE), which we shall now define.

For a given sample of size N, with true outcomes y_i and predicted outcomes $\hat{y}_i$, $i = 1, 2, ..., N$, we define the following quantities:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\text{RSS}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

We consider these measures to be model neutral, and thus suitable for comparing performance between the considered models in this thesis.

4.3.2 Data Pre-Processing

Data pre-processing is considered important in both traditional linear time series forecasting and in statistical learning. Tsay (2010, pp. 83) [6] suggest log transformation for stabilizing variability of price series, which is beneficial for meeting linear model assumptions. Furthermore, the ARMA model relies on

the weak stationarity property of the time series. This is assumed to hold for the differenced log price series, that is, the log return series r_t, which will be considered for both models in this thesis.

4.4 Modeling

4.4.1 ARMA

4.4.1.1 Data Properties

We now only consider the training set and first address the weak stationarity assumption of the log return series. Analysing the first four years in Figure 2, we make similar observations mentioned in section 4.1. ADF unit-root test on training data yield a p-value of 0.01, and we reject the null hypothesis of existance of a unit-root. We assume that the weak stationarity assumption holds for training data.

We continue the analysis by examining the dynamic dependence if log returns. Visual assessment of the sample ACF and PACF in Figure 12 in Appendix B indicates weak serial correlation, however presence of significant serial correlations is indicated at lags 4, 10 and 24. The Ljung-Box test, using $\log(N_{train}) = m_{train} = 7$ lags, gives a p-value of 0.002 which suggests significant serial correlation. This indicates that the log return series can be used in prediction purpose.

4.4.1.2 Order Determination

The next step is to determine the order of the ARMA model, that is, deciding the values of the parameters p and q. To this end, we use the sample ACF and PACF in Figure 12 in Appendix B, to decide reasonable upper limits P and Q, such that $0 \leq p \leq P$ and $0 \leq q \leq Q$. The Cartesian product of these parameter sets define the set of possible ARMA(p,q) models. For each model, generalization performance is estimated by calculating the AICc score, where the lowest scoring model will initially be selected. Residual diagnostics is then performed on this model, as well as nearby models, in order to select the final model.

Based on the ACF and PACF, we choose $P = Q = 10$. Fitting these models using Maximum Likelihood, we conclude that ARMA$(2, 2)$ yields the lowest AICc. The same model is selected by AIC criteria, but BIC only selects the mean model ARMA(0,0).

Before making our final selection, we perform model checking in order to detect possible adequacy of the fitted model. If the model is correctly specified, the model residuals should follow a Gaussian white noise. To check how close the residual series is a white noise, we calculate the Ljung-Box test statistic $Q(m_{train})$. If the residual series is a white noise, then the test statistic follows asymptotically a χ^2 distribution with $m_{train} - g_{train} = 7 - 4 = 3$ degrees of freedom, where $g_{train} = p + q$. The statistic takes a value of 0.3213 and

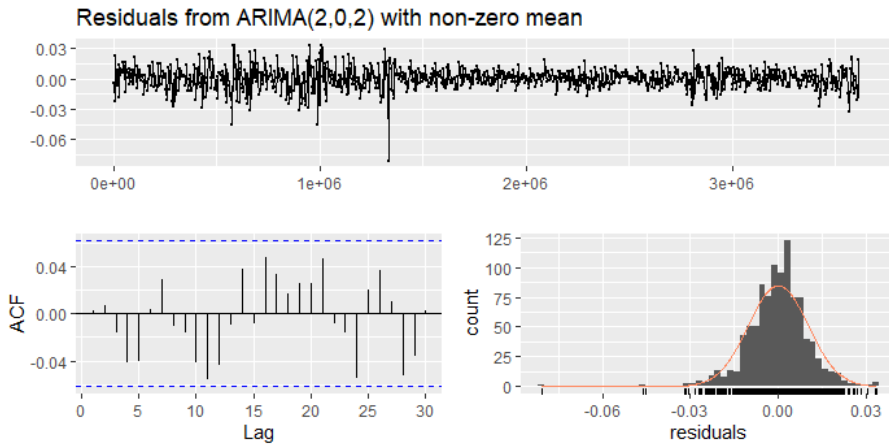

Figure 5: Model checking summary of ARMA(2,2) fitted to the training set.

has a corresponding p-value of 0.956. This suggests we cannot reject the null hypothesis that the residual series is a white noise series.

A visual analysis of the residuals is presented in Figure 5. We note that there are no significant ACFs. The fit to the normal distribution indicates that the normal distribution of the residuals is questionable. This is also indicated by the tail deviations in the Q-Q plot in Figure 13 in Appendix B. Neighbour models do not indicate better fit, thus we accept ARMA$(2, 2)$

$$r_t = 0.000065 - 0.9978r_{t-1} - 0.7361r_{t-2} + a_t - 0.9491a_{t-1} - 0.6465a_{t-2} \quad (32)$$

as the final model.

4.4.2 XGBoost

4.4.2.1 Parameter Tuning Strategy

Some academic literature consider the problem of tuning meta-parameters of XGBoost, but to the best of our knowledge, they are few and none consider specifically XGBoost in the setting described in section 3.6. Luo (2016) [30] provides a general overview of tuning strategies, whilst Probst et al.(2018) [22] implement random search (Bergstra & Bengio, 2012) [31] to assess the tuneability of several common machine learning algorithms, including XGBoost. The choice of tuning strategy in this thesis is influenced by above mentioned academic papers, combined with ideas from Machine Learning and Data Science communities such as Kaggle.

The base of the tuning strategy implemented in this thesis is grid search, which is one of the most commonly used strategies (Bergstra et al., 2012, pp. 281) [31]. The performance of each model setting is estimated using k-fold cross-calidation, and the parameters are tuned in several steps.

Each tuning strategy includes a trade-off between computational speed and accuracy, since the number of models to fit and estimate performance for model selection, grow in a multiplicative manner. More formally, it equals the cardinality of the Cartesian product of the parameter sets. Thus, having a grid generated by several parameter sets does in general mean you are forced to include relatively few values for each parameter. The stepwise grid search approach used in this thesis reduces the number of models, in comparison to a standard grid search, and thus allows for larger parameter sets, but does so at the expense of not taking into account possible dependencies between some parameters. We know from Friedman (2001) [10], that the number of trees $\mathcal{M}$ and learning rate η are dependent and should be determined simultaneously. The number of values tested for each parameter is chosen with aspect of available computational power, while the specific values (see Table 6 in Appendix A) are influenced by Probst et al. (2018) [22]. The initial parameter values are default in the `xgboost` package in R, except for the learning rate η and the number of trees $\mathcal{M}$. The learning rate was chosen such that the number of trees, decides by cross-validation, entailed a reasonable computational load.

In addition to the XGBoost parameters, the supervised learning setting to time series forecasting introduces a window size parameter $\mathcal{W}$. We choose to fix this parameter to $\max(P, Q)$, where P and Q are the maximum values of the ARMA parameters in section 4.4.1. In this way, ARMA and XGBoost both are able to take into account the same number of lagged values. We then rely on the regularization properties of XGBoost to distinguish the relevant features.

The tuning steps are described below, and the actual tuning process is summarized in Table 2.

- **Step 1:** Set the learning rate η and determine the corresponding number of trees $\mathcal{M}$ using cross-validation.
- **Step 2:** Tune the tree-specific parameters, that is, Maximum Tree Depth J, Minimum Child Weight ω, using grid search cross-validation.
- **Step 3:** Update the optimal number of trees $\mathcal{M}$, using cross-validation.
- **Step 4:** Tune randomization parameters, that is, Row Subsample Fraction δ_r and Column Subsample Fraction δ_c, using grid search cross-validation.
- **Step 5:** Tune Regularization parameters, that is, l_1 Regularization Term α and l_2 Regularization Term λ, using grid search cross-validation.
- **Step 6:** Fine tune Learning Rate η and Number of Trees $\mathcal{M}$ by lowering η and finding the corresponding optimal $\mathcal{M}$ using cross-validation.

4.4.2.2 Model Checking

Having found the optimal parameters from the tuning strategy, we fit the model to the entire training set.

XGBoost Meta-Parameter	Initial	Stepwise tuning progress						Final model
Number of trees $\mathcal{M}$	-	$\Rightarrow$ 130		$\Rightarrow$ 130			$\Rightarrow$ 1379	1379
Learning rate η	0.05						$\Rightarrow$ 0.005	0.005
Maximum Tree Depth J	6		$\Rightarrow$ 3					3
Minimum Child Weight ω	1		$\Rightarrow$ 1					1
Row Subsample Fraction δ_r	1				$\Rightarrow$ 0.5			0.5
Column Subsample Fraction δ_c	1				$\Rightarrow$ 1			1
l_1 Regularization Term α	0					$\Rightarrow$ 0		0
l_2 Regularization Term λ	0					$\Rightarrow$ 1		1
Cross-validation mean RMSE	-	1.081	1.054	1.054	1.045	1.033	1.029	1.029

Table 2: Summary of tuning the XGBoost meta-parameters using stepwise grid search cross-validation on the test set.

Figure 6 shows how cross-validation train and test mean RMSE evolve during 5-fold cross-validation in Step 6 of the tuning process, with $\eta = 0.005$. The red dot illustrates the chosen iteration $\mathcal{M} = 1379$. We can see that the test mean RMSE has levelled out at this point, indicating that the chosen $\mathcal{M}$ is close to optimal.

According to section 3.6.1, we shall evaluate whether 5-fold cross-validation is valid for model evaluation for model selection. For this purpose, we shall study the model residuals on training data. Figure 7 shows a summary of properties for the actual model residuals for the final XGBoost model fit to the entire training set. The sample ACFs for the first 10 lags are weak, and not significant. This indicates that the model has been able to capture the linear dependencies input data. To check how close the residual series is a white noise, we calculate the Ljung-Box test. Since we have a window size of $max(P, Q) = 10$, we make a maximum reduction of the degrees of freedom ($max(m_{train} - max(P+Q), 1) = 1$) in the χ^2 distribution for which we assume the test statistic asymptotically shall follow. We get $Q(1) = 1.219$, which corresponds to a p-value of ≈ 0.27. This suggest that we cannot reject the null hypothesis that the residual series is a white noise, assuming common statistical confidence levels of 0.95 or 0.9 are applicable. According to section 3.6.1 this indicates that 5-fold cross-validation is valid for estimating test error of XBGoost model on this specific dataset. The sample distribution indicates the normal assumption of the residuals is questionable.

4.5 Model Comparison (Result)

In this section we will compare how the two above specified ARMA and XGBoost models perform in one-step ahead forecasts on our test set, without refitting.

The test set contains 250 observations of simple log returns of OMX Stockholm Large Cap PI during the period 2019-01-02 to 2019-12-30. In accordance with traditional time series forecasting, we will use the last (ordered) observations from the training set in order to construct full feature vectors for the first response value in the test set. This means that, for the first predicted value, XGBoost will use the 10 last observations from the training set, while ARMA$(2, 2)$ will use the 2 last observations from the training set.

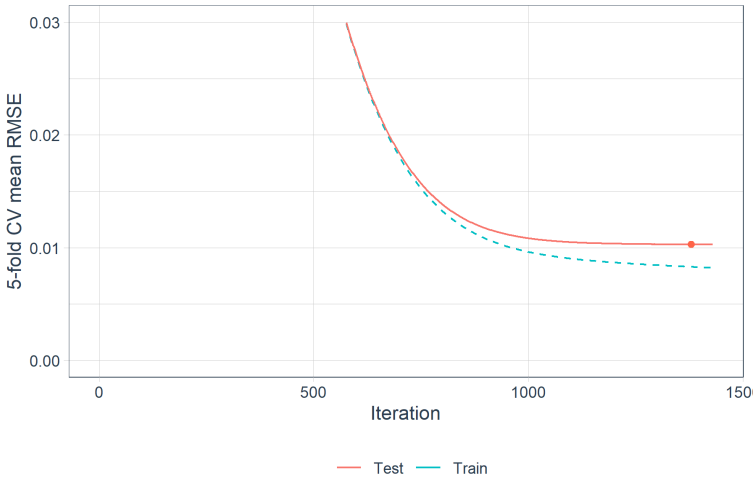

Figure 6: Development of train and test mean RMSE from 5-fold cross-validation during Step 6 of the XGBoost parameter tuning process, having fix learning rate $\eta = 0.005$. The red dot illustrates the chosen number of iterations $\mathcal{M} = 1379$.

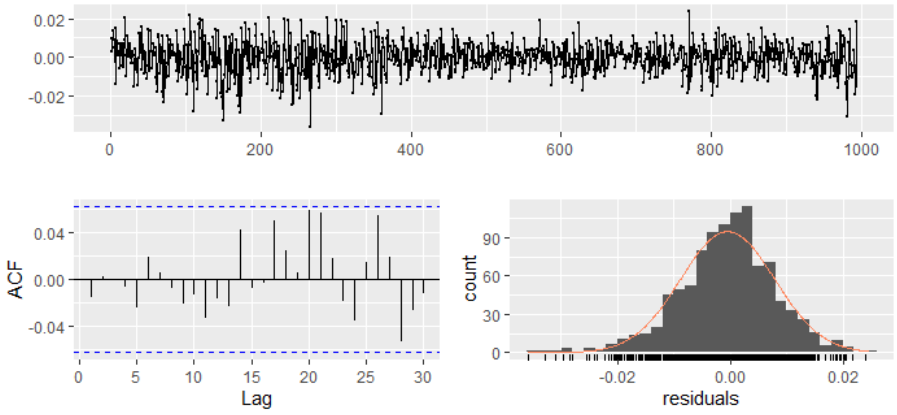

Figure 7: Summary of model residuals for the final XGBoost model.

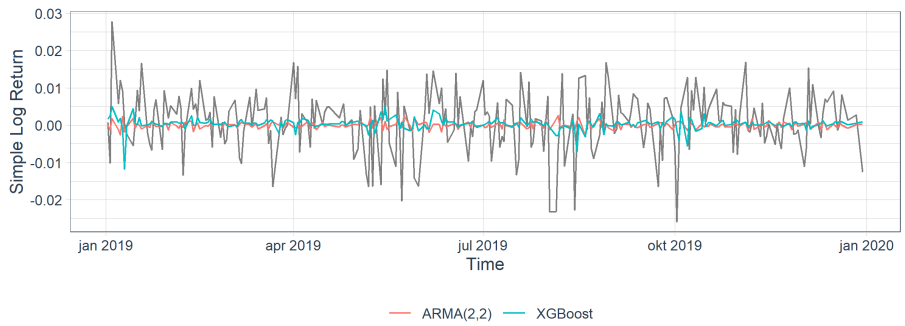

Figure 8: Test set simple log returns against predicted log returns from ARMA and XGBoost models.

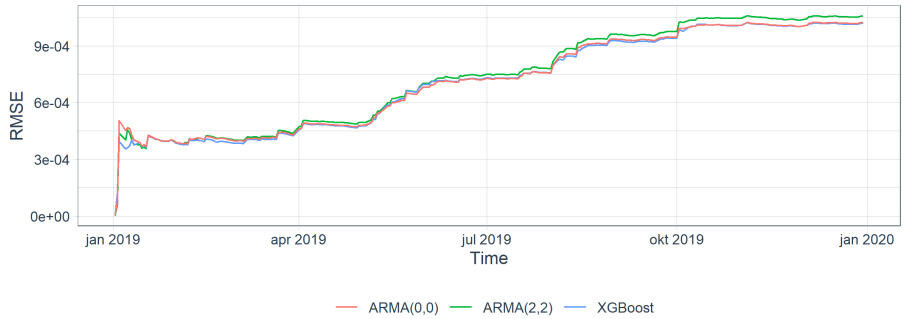

Figure 9: Development of RMSE for one-step ahead test set forecasts from ARMA and XGBoost models.

4.5.1 One-Step Ahead Forecast

Figure 8 illustrates the one-step ahead log return forecasts against observed logr returns over the test set, for both models, while figure 9 shows the development of the corresponding RMSE, where a mean model ARMA(0,0) is also included. In the first figure, we can see that both models tend to underestimate the magnitude of the log returns. XGBoost shows more fluctuations than ARMA(2,2), in which predictions show mostly small deviations from its intercept. The second figure illustrates how RMSE corresponding to XGBoost and ARMA(2,2), both follows the RMSE developement of the mean model. At the end of the test period, the ARMA(2,2) model has the largest RMSE, slightly above XGBoost and the mean model which shows very similar results.

Although regression is the main interest of this thesis, we have also compared the above three models in terms of predicting whether the one step log return will be positive or negative. We simply used the log returns above and converted them to observations of a binary classification problem. Table 3 and

4 show the corresponding confusion matrices for ARMA and XGBoost respectively. Test data contains 149 observed positive log returns and 101 negative. The ARMA model predicted as many positives as negatives, while XGBoost predicted just under 70 % positive. Common classification performance measures corresponding to above confusion matrices are given in Table 5.

	Actually positive	**Actually negative**	
Predicted positive	73	52	125
Predicted negative	76	49	125
	149	101	

Table 3: Confusion matrix on the test set for the ARMA(2,2) model.

	Actually positive	**Actually negative**	
Predicted positive	109	65	174
Predicted negative	48	32	80
	149	101	

Table 4: Confusion matrix on the test set for the XGBoost model.

	Regr. measures	**Classif. measures %**		
	RMSE	**Sens.**	**Spec.**	**Acc.**
XGBoost	0.001023	73.2	35.6	56.4
ARMA(2,2)	0.001060	49.0	48.5	48.8
ARMA(0,0)	0.001026	100	0	59.6

Table 5: Summary of performance measures for XGBoost, ARMA(2,2) and ARMA(0,0) on the test set.

5 Discussion

This thesis has examined the suitability of XGBoost in problems regarding prediction of univariate financial time series. This included partly deriving a theoretical background in financial time series analysis, XGBoost in a regression setting and a supervised learning setting for time modelling of such time dependencies, and partly applying this theory in a case study on five years of daily market index data. In this section we will discuss the findings of this study.

The first two sections are dedicated to the theoretical conditions of XGBoost, while the third section concerns data specific findings from the case study. Some suggestions for further studies are addressed in section four, and finally the main conclusions of this study are made in the fifth section.

5.1 Theory

In answering the question whether XGBoost is a suitable model for predicting univariate financial time series, one of the main issues concerned if the supervised learning setting of the one step-ahead time series forecasting problem, which among others is mentioned in Bontempi et al. (2013, sect. 3.1) [1], is theoretically motivated. This included dealing with contradictions regarding the assumption of independent observations in conjunction with model selection and assessment. In section 3.6 the supervised learning setting in conjunction with non-modified k-fold cross-calidation, were assumed to hold for XGBoost using findings in the paper by Bergmeir et al. (2017) [5]. The paper does not specifically mention XGBoost or tree-based methods, but rather states that this approach holds for pure autoregressive models which includes Machine Learning procedures for time series forecasting. Bergmeir et al. state it is the first paper suggesting non-modified k-fold cross-calidation in the dependent case, which indicates there is still a lot of work to be done in this area.

Hastie et al. (2009, pp. 313) [8] state that trees has difficulty in capturing additive structures. Assuming boosting trees does not solve this issue, this also includes XGBoost. This needs to be handled appropriately if the time series of interest can be expressed as an additive decomposition $Y_t = S_t + T_t + \varepsilon_t$, where S_t is a seasonal component, T_t is a trend and ε is an error term (Hastie et al, 2009, pp. 297). We will discuss this situation to some extent in section 5.3.

Under the assumption that we have indeed a valid supervised learning setting to the univariate time series forecasting problem, we can proceed with theoretical findings regarding whether XGBoost can be considered an appropriate model for this type of problem. The success in predictive problems of boosted tree methods in general is well documented, both through empirical studies and motivated theoretically in for example Friedman et al. (2000) and Friedman (2001, 2002) [10, 11]. During the derivation of XGBoost, we have not found any reason to believe that the predictive capability would not hold in the time series setting. We will however mention a couple of properties. Firstly, tree-based methods in general, and thus also XGBoost, naturally deal with collinearity in the tree structure procedure. Since the split criteria is evaluated in terms of gain, it is likely that only one of these will be selected for a split, while further splits on the remaining correlated features will have little or no gain in terms of prediction performance. However, Strobl et al. (2008, sect. 2.2) [32] suggest that the correlated features are used interchangeably in the tree building process. Thus the most influential of the correlated variables may not be selected for the split, which has an effect on interpreting the contribution of each variable through feature importance. Secondly, regression trees are unable to extrapolate, since the response corresponds to an average of training data

observations.

5.2 Case Study

In this section we will discuss the content in section 4, which includes choice of methods related to the application of XGBoost in a supervised learning setting to univariate time series forecasting, the experiment setup and interpretation of the result.

5.2.1 Linear Time Series Background

In order to evaluate the performance of XGBoost, we chose to compare to the much established and relatively simple linear ARMA model. To make a fair comparison, we chose to use the same format on input data for both models. The pre-processing and analysis of the data where highly influenced by theory regarding analysis of linear time series, of financial type. This entailed selecting modelling of log returns, to facilitate the weak stationary assumption. According to Ahmed et al. (2010, sect. 3) [2] differentiation of time series may have negative effect on predictive performance for non-linear models, such as XGBoost. However, since XGBoost using regression-trees as weak learners are unable to extrapolate, the pre-processing methods must take this into account. We find that the weak stationary assumption of input data avoids the need for extrapolation. Furthermore, Hastie et al (2009) [8] suggest that standardization of variables is beneficial for ridge regression, and thus may also apply for XGBoost with respect to the penalized learning objective. We discuss alternative pre-processing approaches to some extent in section 5.3.

5.2.2 Data

In this section we will discuss the choice of dataset, properties of the dataset at hand and how this may have influenced the result.

The descriptive statistics of the log return series and return series of OMX Stockholm Large Cap PI, summarized in Table 1, reveal little difference between these series. Several measures indicate analyzing the log transformed series has little stabilizing effect on the specific dataset. Statistical tests of sample skewness and kurtosis indicated that the normal assumption of log returns does not hold. This may effect the performance of the ARMA model, which assumes normality in the maximum likelihood estimation (Tsay, 2010, pp. 19) [6]. The XGBoost model however makes no distributional assumption of data.

The sample ACFs and PACFs for the complete dataset are given in Figure 4. Although statistical tests indicate significant autocorrelation, it can be seen that the individual correlations are weak. This implicates (Tsay, 2010, pp. 36) [6] that the log return series of OMX Stockholm Large Cap PI, during the period 2015-2019, is close to a white noise series. The same goes for the period 2015-2018 which constitutes the training data, and it is likely to hold true also for the test data, although this has not been tested.

Figure 2 reveals some obvious outliers in the dataset, which has been kept in the study. Although these types of observations occur from time to time, they are often consequences of exceptional events. It would be reasonable to exclude these events from the dataset to obtain a more stable series, which should benefit the normal assumption, and have a positive effect when using least squares loss criteria for XGBoost (Hastie et al., 2009, pp. 349-350) [8].

5.2.3 Modeling

Since XGBoost is the model of primary interest in this thesis, we will only do a brief analysis of the content related to the ARMA modelling in the succeeding paragraph, to dedicate the remaining part of this section to XGBoost.

For the ARMA model, model selection depends on the choice of information criteria. Both AIC and AICc select the ARMA(2,2) model, while BIC selects the mean model ARMA(0,0). This is not surprising, considering the weak autocorrelations in data, and that BIC by definition penalizes complex models to a greater extent than AIC and AICc. Model checking, including test of autocorrelation of the model residuals, indicates that the model is reasonable specified, although Figure 5 and 13 indicate the normal assumption of the residuals is questionable. The latter should not be surprising, given that the normal assumption of the log return series does not hold.

5.2.3.1 Parameter tuning

Table 2 shows how the model parameters and the cross-validation mean test RMSE developed in each step of the parameter tuning process. We note how the mean test RMSE decreases for each step, except in the third step since the number of iterations or trees does not change.

Figure 10 depicts how the cross-validation mean RMSE developed when fine tuning the learning rate and the number of iterations in the final step of the parameter tuning process. It is here shown how decreasing the learning rate entails slow learning which increases the optimal number of iterations. The test curves, corresponding to the three largest learning rates, all show only a small increase after reaching the minimum value, and then level off to be near constant as the number of iterations increases. This result is common in many applications (Hastie et al., 2009, pp. 371) [8], and indicates that over-fitting is not a major concern for any of these models, but it is necessary to choose the number of iterations sufficiently large to avoid under-fitting. For the model with the smallest learning rate, there is not enough observations to fully evaluate the model. However, the curve looks to have flattened out at 7000 iterations, and since the mean RMSE is above the minimum for the final model, it is reasonable to assume that choosing a learning rate below 0.005 would not yield significantly improved model performance.

Model checking, including statistical tests of autocorrelations of the model residuals, indicates that the model has been specified in a reasonable manner.

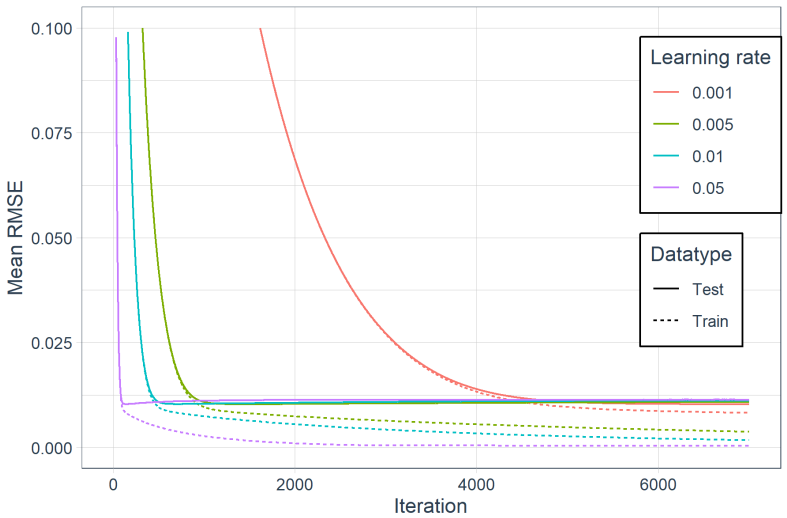

Figure 10: Cross-validation mean RMSE as a function of the number of iterations for different learning rates in the tuned XGBoost model.

As mentioned in section 4.4.2, the chosen parameter tuning strategy may have some weaknesses. This includes not taking all possible dependencies of all the meta-parameters into consideration during the fit, as a full grid search cross-validation strategy would. However, we believe the tuning strategy is solid enough to give a reasonable fit.

5.2.4 Model Interpretation

Observing the final meta-parameters of the XGBoost model, we do not notice any obvious surprising results. The shallow trees indicates little interaction between the features, which should not be surprising given the weak dependencies in the log return series.

The implementation of XGBoost in R offers the functionality of assessing feature importance (see sect. 3.5.5), which for the final XGBoost model is summarized in Figure 11. The Gain relate to loss reduction when using a feature for splitting, while Cover and Frequency are related to the number of times a feature is used to split the data across all trees. A basic evaluation of the three plots shows how the features corresponding to lag 1, 2, 3, 4 and 10 seem to contribute more than features 5-9. This result may not be surprising, since time series analysis theory suggests (Tsay, 2010, pp. 105) [6] that the most recent past values are most likely to be important in forecasting purpose. Furthermore, the sample ACF and PACF on the training set, visualized in Figure 12 in Appendix B, may explain the importance of features 4 and 10. At the same time, relatively low ACF and PACF for features 1 and 2 mIay not explain why these are considered top contributing features in the XGBoost model. However,

(a) (b) (c)

Figure 11: Feature importance summary of Gain, Cover and Frequency for the final XGBoost model.

remember that ACF and PACF only measure linear dependencies, whilst XGBoost is able to capture more complex relationships. These results may indicate that the XGBoost model can indeed capture the dynamic structure of the time series, which we find to be very interesting.

5.2.5 Case Study Performance Evaluation

The results of the case study are presented in Figure 8 and 9. Here we can clearly see that neither XGBoost nor ARMA(2,2) predicts the movements of the log returns during the test period in a satisfactory manner. The magnitude of the true log returns are severely underestimated by both models. Thus, in Figure 9, it can be seen how the RMSE increases significantly during relatively volatile periods of the test period. The ARMA(2,2) model and the XGBoost model perform only marginally better than the mean model ARMA(0,0), and we conclude there is little gain in using more complex models than the mean model on the specific dataset. The poor performance of the predictions is expected, given the weak dependencies in the dataset, as mentioned in above sections.

Although regression was the primary task, we also examined model performance in terms of binary classification, that is, the models' ability to predict positive or negative returns. The results are summarised in Table 3, 4 and 5. We note that the XGBoost model predicted a larger proportion of positive log returns in comparison to the ARMA(2,2) model which seems to predict similar to random guessing. The XGBoost model shows better sensitivity, but worse specificity than the ARMA(2,2) model. Taking overall accuracy into consideration, the mean model performs best and we conclude that neither XGBoost nor ARMA(2,2) perform satisfactory binary classification. It seems like XGBoost, to some extent, can estimate the proportion of positives and negatives, but it can not predict which days will have one or the other.

5.3 Suggestions for Further Studies

We realize that this study only considers the supervised learning setting to time series forecasting and application of the XGBoost in a very non-profound matter,

considering the wide range of theories on the subject that we have encountered during the writing of this thesis. We end this section by mentioning some of our main ideas for further studies.

For empirical evaluation of XGBoost in the problem at hand, or any other method, it is of course necessary that the dataset is considered predictable. Considering the Efficient Market Hypothesis (Bechelier, 1900) [33], such series does not exist on the financial market. However, this hypothesis has met some criticism. Malkiel (2003) [34] concludes that pricing irregularities and predictive patterns in stock returns can appear over time and persist for short periods, and that the market cannot be perfectly efficient. This would implicate that further studies of time series, similar to that in this thesis, can be used to evaluate the XGBoost model in a supervised learning setting to forecasting time series.

For empirical evaluation of XGBoost in a wider spectrum of time series, a similar approach to that in Ahmed et al. (2010) [2], in which the M3-competition dataset is used, could be attempted. Such approach may be beneficial in comparison to other models, since this dataset has frequently been used in many studies of which the results are available online.

Feature engineering, is widely accepted to be an important aspect regarding performance of machine learning models. This thesis leaves room for increasing performance of the XGBoost model by considering optimizing the format of the input data to a greater extent.

Modeling of seasonal, cyclic or trend (for non-stationary time series) components has not been considered in this thesis. However, some experimental examination of deseasonalization, assuming an additive decomposition[5], of the price series was performed. A similar deseasonalization approach is applied in Ahmed et al. (2010, section 5) [2], which also refers to several papers considering the issues of pre-processing data. For the specific dataset, deseasonalization seemed to have some positive effect on stabilizing the log return series which may be beneficial for model performance.

In general, forecasting the remainder of a decomposed time series, to some extent described in Hyndman & Athanasopoulos (2018, sect. 6.7-8) [35], may be one method to overcome the issues of XGBoost using regression trees being unable to extrapolate. Possibly also to overcome that regression trees themselves cannot capture additive structures (Hastie et al., 2009, pp. 313) [8]. Another interesting decomposition approach worth mentioning is the so called CEEMDAN-XGBOOST used in Yingrui et al.(2019) [36], where XGBoost is applied to a CEEDMAN-decomposition of historical crude oil prices.

It is likely that the model selection procedure of XGBoost can be improved, by choosing other types of cross-validation based methods mentioned in for example Bergmeir et al. (2012, 2017, 2020) [25, 5, 28], and Hyndman & Khandakar (2008) [27]. This may also influence how the division of test and training set is performed, and whether it is reasonable to re-fit the model during each step when assessing model performance.

[5]We used the function `stats::decompose` in R to decompose training data.

5.4 Conclusions

The main aim of this thesis was to assess the suitability of XGBoost on one-step ahead forecasting of univariate time series. In chapter 3, we derived the XGBoost model and a framework for phrasing univariate time series forecasting as a supervised learning problem. We can conclude that the XGBoost model is indeed a powerful tool, which has some properties that make it a candidate method for application in prediction of univariate time series. Such properties are its explicit and implicit ability perform feature selection and to handle multicollinearity, which may cause issues in regression problems in general. Feature importance facilitates interpreting the contribution of each predictor, which we understand may be an important aspect in time series forecasting.

We have seen that its flexibility in terms of the many parameters complicates the model selection process in which the choice of an appropriate tuning strategy is key for successful prediction. Applying XGBoost in a time series forecasting setting adds more complexity to the model assessment process, due to the ordered and dependent nature of time series. Time series forecasting theory provides several cross-validation based assessment methods which are intended to take into account the properties of time series, as alternatives to the standard k-fold cross-calidation assessment with a 70/30 random partitioning. Both theory and empirical studies suggest that such special methods will not necessarily outperform k-fold cross-calidation, and that the validity of an assessment method should be evaluated for the problem at hand.

The result of the case study in section 4.5 shows that neither XGBoost nor ARMA(2,2) significantly outperform a mean model on the specific dataset. With regards to the concept of parcimony (Tukey, 1961) [37], we make the assessment that the mean model is the best suited model on the specific dataset. The results on the specific dataset may largely be explained by the fact that the efficient market hypothesis applies, and it is not likely that any model would significantly outperform a mean model. However, feature importance of XGBoost shows that features of relatively high autocorrelation are among the top contributors to the prediction, together with the most recent observations. This is in line with traditional time series theory. Although the most recent observations did not show significant autocorrelation, there may exist more complex relationships that XGBoost can capture.

Our perception is that much is unexplored when it comes to applying supervised learning methods on time series forecasting problems in general. If we further limit ourselves to XGBoost in this setting, the availability of academic work is almost non-existent. However, we are under the impression that interest in this area is increasing, and success of XGBoost in machine learning competitions may motivate research on this specific method.

References

[1] *G. Bontempi, S. Ben Taieb, Y.-A. Le Borgne, Machine Learning Strategies for Time Series Forecasting, Vol. 138, 2013.*

[2] *N. Ahmed, A. Atiya, N. Gayar, H. El-Shishiny, An empirical comparison of machine learning models for time series forecasting, Econometric Reviews 29 (2010) 594–621.*

[3] *T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16 (2016).*

[4] *F. Sigrist, Gradient and newton boosting for classification and regression (2019).*

[5] *C. Bergmeir, R. J. Hyndman, B. Koo, A note on the validity of cross-validation for evaluating autoregressive time series prediction, Computational Statistics & Data Analysis 120 (2018) 70 – 83.*

[6] *R. S. Tsay, Analysis of Financial Time Series, 3rd Edition, Wiley Series in Probability and Statistics, John Wiley and Sons, 2010.*

[7] *G. E. P. Box, et al., Time Series Analysis: Forecasting and Control, 5th Edition, Wiley Series in Probability and Statistics, Wiley, 2015.*

[8] *T. Hastie, R. Tibshirani, J. H. Friedman, The Elements of Statistical Learning, 2nd Edition, Springer Series in Statistics, Springer, 2009.*

[9] *K. P. Murphy, Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.*

[10] *J. H. Friedman, Greedy function approximation: A gradient boosting machine, The Annals of Statistics 29 (5) (2001) 1189–1232.*

[11] *J. H. Friedman, Stochastic gradient boosting, Computational Statistics and Data Analysis 38 (4) (2002) 367–378.*

[12] *G. Ljung, G. Box, On a measure of lack of fit in time series models, Biometrika 65 (1978).*

[13] *J. Y. Campbell, A. W. Lo, A. MacKinlay, The Econometrics of Financial Markets (1997).*

[14] *R. S. Tsay, G. C. Tiao, Consistent estimates of autoregressive parameters and extended sample autocorrelation function for stationary and non-stationary arma models, Journal of the American Statistical Association 79 (385) (1984) 84–96.*

[15] *H. Akaike, A new look at the statistical model identification, IEEE Transactions on Automatic Control AC-19 (6) (1974) 716–723.*

[16] *N. Sugiura, Further analysts of the data by akaike' s information criterion and the finite corrections, Communications in Statistics - Theory and Methods 7 (1) (1978) 13–26.*

[17] *G. Schwarz, Estimating the dimension of a model, The Annals of Statistics 6 (2) (1978) 461–464.*

[18] *K. P. Burnham, D. R. Anderson, Multimodel inference: Understanding aic and bic in model selection, Sociological Methods & Research 33 (2) (2004) 261–304.*

[19] *L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees, Wadsworth and Brooks, 1984.*

[20] *Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences 55 (1) (1997) 119 – 139.*

[21] *J. H. Friedman, Additive logistic regression: A statistical view of boosting, The Annals of Statistics 28 (2) (2000) 337–407.*

[22] *P. Probst, A.-L. Boulesteix, B. Bischl, An analysis of transformations, Journal of Machine Learning Research (20) (2019) 1–32.*

[23] *T. Chen, et al., Extreme Gradient Boosting, Vienna, Austria (2020).*
URL `https://cran.r-project.org/web/packages/xgboost/xgboost.pdf`

[24] *E. Zivot, J. Wang, Modeling Financial Time Series with S-PLUS®, Springer-Verlag, 2006.*

[25] *C. Bergmeir, J. M. Benítez, On the use of cross-validation for time series predictor evaluation, Information Sciences 191 (2012).*

[26] *J. D. Hart, Automated kernel smoothing of dependent data by using time series cross-validation, Journal of the Royal Statistical Society Series B (Methodological) 56 (1994).*

[27] *R. J. Hyndman, Y. Khandakar, Automatic time series forecasting: the forecast package for R, Journal of Statistical Software 26 (3) (2008) 1–22.*

[28] *R. J. Hyndman, et al., forecast: Forecasting functions for time series and linear models, r package version 8.12 (2020).*
URL `http://pkg.robjhyndman.com/forecast`

[29] *H. Wickham, et al., Welcome to the tidyverse, Journal of Open Source Software 4 (43) (2019) 1686.*

[30] *G. Luo, A review of automatic selection methods for machine learning algorithms and hyper-parameter values., NetMAHIB 5 (1) (2016) 18.*

[31] *J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research 13 (10) (2012) 281–305.*

[32] *C. Strobl, A.-L. Boulesteix, T. Kneib, T. Augustin, A. Zeileis, Conditional variable importance for random forests, BMC Bioinformatics 9 (1) (2008) 307.*

[33] *L. Bachelier, Théorie de la spéculation, Annales Scientifiques de L'Ecole Normale Supérieure 17 (1900) 21–88, reprinted in P. H. Cootner (ed), 1964, The Random Character of Stock Market Prices, Cambridge, Mass. MIT Press.*

[34] *B. G. Malkiel, The efficient market hypothesis and its critics, Journal of Economic Perspectives 17 (1) (2003) 59–82.*

[35] *G. Athanasopoulos, R. J. Hyndman, Forecasting: Principles and Practice, 2nd Edition, 2018.*

[36] *Y. Zhou, T. Li, J. Shi, Z. Qian, A CEEMDAN and XGBOOST-Based Approach to Forecast Crude Oil Prices, Complexity (2019) 1–15.*

[37] *J. W. Tukey, Discussion, emphasizing the connection between analysis of variance and spectrum analysis, Technometrics 3 (2) (1961) 191–219.*

Appendix

A Tables

XGBoost Parameter	Initial	Parameter Set
Number of trees $\mathcal{M}$	130	-
Learning rate η	0.05	$\{0.05, 0.01, 0.005, 0.001\}$
Maximum Tree Depth J	6	$\{3, 5, 6, 7, 9, 11, 13, 15\}$
Minimum Child Weight ω	1	$\{1, 3, 5, 7\}$
Row Subsample Fraction δ_r	1	$\{0.5, 0.6, ..., 1\}$
Column Subsample Fraction δ_c	1	$\{0.4, 0.6, ..., 1\}$
l_1 Regularization Term α	0	$\{0, 0.001, 0.01, 0.1, 0.3, 0.5, 1, 2, 5\}$
l_2 Regularization Term λ	0	$\{0, 0.001, 0.01, 0.1, 0.5, 1, 2, 5, 10, 20, 30\}$

Table 6: XGBoost parameter values.

B Figures

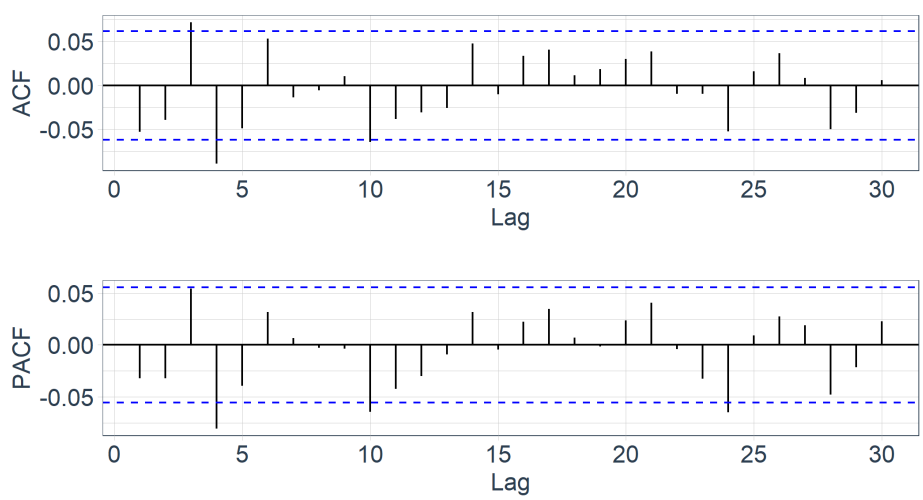

Figure 12: Sample ACF and PACF for the training set of daily OMX Stockholm Large Cap PI.

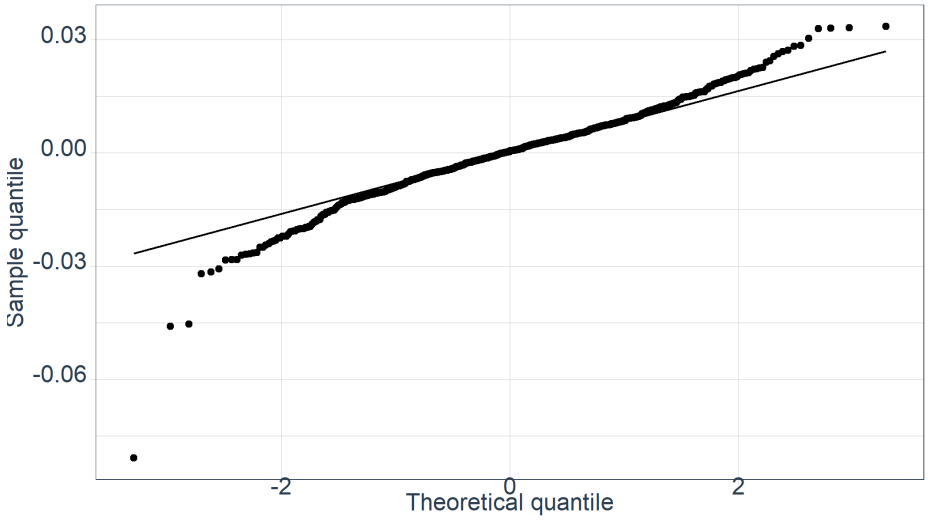

Figure 13: Q-Q plot of residuals for ARMA(2,2) fit to train set