PERFORMANCE OF HYBRID MACHINE LEARNING ALGORITHMS ON FINANCIAL TIME SERIES DATA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MERVE GÖZDE SAYIN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
FINANCIAL MATHEMATICS

FEBRUARY 2021

Approval of the thesis:

PERFORMANCE OF HYBRID MACHINE LEARNING ALGORITHMS ON FINANCIAL TIME SERIES DATA

submitted by **MERVE GÖZDE SAYIN** in partial fulfillment of the requirements for the degree of **Master of Science in Financial Mathematics Department, Middle East Technical University** by,

Prof. Dr. A. Sevtap Selçuk Kestel _____
Director, Graduate School of **Applied Mathematics**

Prof. Dr. A. Sevtap Selçuk Kestel _____
Head of Department, **Financial Mathematics**

Assoc. Prof. Dr. Ceylan Yozgatlıgil _____
Supervisor, **Department of Statistics, METU**

Prof. Dr. Ömür Uğur _____
Co-supervisor, **Institute of Applied Mathematics, METU**

Examining Committee Members:

Prof. Dr. A. Sevtap Selçuk Kestel
Institute of Applied Mathematics, METU _____

Assoc. Prof. Dr. Ceylan Yozgatlıgil
Department of Statistics, METU _____

Prof. Dr. Ömür Uğur
Institute of Applied Mathematics, METU _____

Prof. Dr. Seher Nur Sülkü
Department of Econometrics, Ankara Hacı Bayram Veli University _____

Assoc. Prof. Dr. Ebru Yüksel Haliloğlu
Industrial Engineering Department, Gazi University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: MERVE GÖZDE SAYIN

Signature :

ABSTRACT

PERFORMANCE OF HYBRID MACHINE LEARNING ALGORITHMS ON FINANCIAL TIME SERIES DATA

Sayın, Merve Gözde
M.S., Department of Financial Mathematics
Supervisor : Assoc. Prof. Dr. Ceylan Yozgatlıgil
Co-Supervisor : Prof. Dr. Ömür Uğur

February 2021, 89 pages

Estimating stock indices that reflect the market has been an essential issue for a long time. Although various models have been studied in this direction, historically, statistical methods and then various machine learning methods have to introduced artificial intelligence into our lives. Related literature shows that neural networks and tree-based models are mostly used. In this direction, in this thesis, four different models are examined. The first one is the most preferred neural network method for financial data called LSTM, and the second one is one of the most preferred tree-based models called XGBoost, and the third and the fourth models are the hybridizations of LSTM and XGBoost. Besides, these models have been applied to the total of nine stock market indexes, three from European markets, three from Asian and three from American markets, and the model that gives the best results is determined according to the Mean Absolute Scaled Error (MASE) evaluation criteria.

Keywords: LSTM, XGBoost, Hybrid Models, Machine Learning, Stock Market Index

ÖZ

HİBRİT MAKİNE ÖĞRENME ALGORİTMALARININ FİNANSAL ZAMAN SERİLERİ VERİLERİ ÜZERİNDEKİ PERFORMANSI

Sayın, Merve Gözde
Yüksek Lisans, Finansal Matematik Bölümü
Tez Yöneticisi : Doç. Dr. Ceylan Yozgatlıgil
Ortak Tez Yöneticisi : Prof. Dr. Ömür Uğur

Şubat 2021, 89 sayfa

Piyasayı yansıtan hisse senedi endeksleri tahminlemesi uzun zamandır süregelen önemli bir tartışma konusudur. Bu doğrultuda çeşitli modeller kullanılmış olsa da tarihsel olarak önce istatistiksel metotlar ve daha sonra yapay zekanın hayatımıza girmesiyle beraber çeşitli makine öğrenmesi metotları denenmiştir. Literatüre göre en çok sinir ağları ve ağaç bazlı modeller kullanılmıştır. Bu doğrultuda ise bu tezde sinir ağları yöntemlerinden finansal veriler için en fazla tercih edilen LSTM ve ağaç bazlı modellerden son zamanların gözdesi olan XGBoost ve bu iki modelin hibritlenmesinden meydana gelen toplam dört model incelenmiştir. Ayrıca bu modeller üçü Avrupa, üçü Asya ve üçü Amerika olmak üzere dokuz farklı hisse senedi endeksi üzerinde uygulanmış ve en iyi sonucu veren model MASE değerlendirme kriterine göre açıklanmıştır.

Anahtar Kelimeler: LSTM, XGBoost, Hibrit Modeller, Makine Öğrenmesi, Hisse Senedi Endeksi

ACKNOWLEDGMENTS

I would like to express my very great appreciation to my thesis supervisor Assoc. Prof. Dr. Ceylan Yozgatlıgil for her patient guidance, enthusiastic encouragement, and valuable advice during the development and preparation of this thesis. Her willingness to give her time and to share her experiences has brightened my path.

Also, I would like to thank my co-advisor, Prof. Dr. Ömür Uğur to give his precious time and to guide me at the beginning of selecting the subject of the thesis.

I deeply thank dear Res. Assist. at IAM Özge Tekin for her academic and mental support.

I would like to individually thank my family, my friends, and my loved ones, but most importantly, I would like to express my great appreciation to all the women who made me who I am today, brought me to this day, entered my life and touched my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

A3C	Asynchronous Actor-Critic Agents
AI	Artifical Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
AR	Autoregressive
ARIMA	Autoregressive Integrated Moving Average
CART	Classification and Regression Trees
DL	Deep Learning
GBDT	Gradient Boosting Decision Trees
LSTM	Long Short-Term Memory
MA	Moving Average
MAPE	Mean Absolute Percentage Error
MASE	Mean Absolute Scaled Error
MCTS	Monte-Carlo Tree Search
ML	Machine Learning
PCA	Principal Component Analysis
RF	Random Forest
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines
SVR	Support Vector Regression
t-SNE	t-Distributed Stochastic Neighbor Embedding
U.S.	United States
WN	White Noise
XGBoost	eXtreme Gradient Boosting

CHAPTER 1

INTRODUCTION

One of the most fundamental instincts of human beings is to know the unknown or predict the future that best suits the relevant situation. Making accurate and successful predictions in finance and all areas of life has become a crucial issue for humanity because it is challenging to take action against the unknown. This situation can create a wide variety of uncertainties. Making decisions amid these uncertainties is an issue that should be considered critically important, especially when it comes to finance. This is why forecasting has been included in all areas of our lives in different subjects thanks to the developing technology. With machine learning techniques developed with artificial intelligence, quite different doors have been opened, and developments continue at full speed.

One of the building blocks of finance, estimating stock market indices, aims to analyse the market much better and steer the market by making the necessary decisions. Most straightforwardly, the data consists of a time column and a value column corresponding to that time, which is called time series. It can be stated that forecasting over time series is one of the most frequently used methods in finance and, of course, in stock market indices, considering the advantage of defining the series's structure in detail. Currently, time series analysis can be performed not only statistically but also by machine learning methods. Unfortunately, the best model has not been selected in the literature yet because each series contains various features, and each algorithm has both advantages and disadvantages. The aim should be to choose the method that can capture the dynamics of the data well so that the produced forecasts can follow the actual behavior of the series. Zhang [90] mentioned that time series should be

modelled with hybrid methods in 2003. He explains the necessity of this situation as follows: Since linear and non-linear parts of time series have different properties, they should be handled in different ways. In his article, residuals are obtained by using the ARIMA model for the linear part. However, if these residuals still have a serial correlation, linear models are no longer good enough to capture this pattern. Furthermore, residual analysis cannot determine the non-linear structure. For this reason, remodelling the residuals obtained from ARIMA with machine learning methods (ANN) should help explore the non-linear structure in the time series. As a result, Zhang suggests using hybrid models for time series in his article.

The motivation of this study comes from the Makridakis Competition which is a competition that is open to everyone on the Kaggle platform, where forecasters are generally interested in developing models, contributing to technology, sharing information, and giving a financial award to those placed in the rankings. The fourth of this competition was held in 2018, and studies on the fifth are recently finished. Like the past three M Competitions, M4 was an open competition to guarantee reasonableness and objectivity. M4 Competition was reported at the starting of November 2017. It was implemented on $100,000$ time series data which include novel features which are high-frequency (weekly, daily, and hourly) and low-frequency (yearly, quarterly and monthly) data, prediction intervals (PIs) and point forecasts (PFs), reproducibility of the outcomes, the inclusion of an endless number of diverse series and bench-marks. The competition ended on May 31st, 2018. A brief paper with the initial outcomes of the M4 was published in the International Journal of Forecasting on June 20th, 2018 [57]. As a satisfactory result of the M4 Competition, there are seven significant findings. However, the most significant one is the ultimate success of hybrid models. In other words, the successful combination of pure statistical and pure ML models gives more accurate results. It can be stated that single models cannot efficiently capture the time series pattern sufficiently in general. But, a specific combination of these models is more successful because they minimize the other's errors by averaging.

In this direction, this thesis aims to examine the results by applying two different machine learning methods and the hybridized model of these two models to time series over nine different stock market indexes providing a contribution to this subject.

Hence hybrid models obtained by two ML algorithms can capture all the dynamics of the series if only one ML method miss to explain all behaviour.

In this chapter, there are three sections. The aim of the thesis and the literature review is given in the first two. The last section clarifies the structure of the thesis.

1.1 Aim of the Thesis

The thesis aims to find the best machine learning algorithm that predicts the stock market index in the best manner. Our research question here is if one model cannot capture all necessary features of the time series, can we use another model to explain the remaining unexplained characteristics of the series. To answer this question we chose two best performed techniques for prediction purposes given in the literature. Predicting stock market index combines more than one discipline; finance, mathematics, and statistics. In the literature, various algorithms and indices are used for predicting the stock market index. In this thesis, only two main algorithms and nine data are used. LSTM and XGBoost algorithms are preferred since they are implemented in several libraries for various programming languages, including R, and many of them are open source. XGBoost is an excellent prediction method in statistics, and LSTM is a modern forecasting method for time series. Moreover, within this thesis it was tried to answer some questions, which are;

- Should any hybrid models be used?
- Does hybridization make results better?
- Does hybridization of LSTM and XGBoost give better results?
- Which hybrid models give the best outcomes?

Answering these questions can contribute to the literature.

These algorithms, LSTM and XGboost and their combinations, have been preferred because they are the most commonly used and recommended algorithms in the literature. Also, they offer better results by playing with hyper-parameters. Furthermore,

working with time series has many advantages such as scaling and usability.

Numerical investigations and the visualizations in this thesis are conducted by using R Language, EViews, Python and Microsoft Excel.

1.2 Literature Review

The model prediction itself is a developed area where most people want to be successful, but the stock prediction is notably a significant problem. About this long-standing problem, Fama [28] states the following with the Efficient Market Hypothesis;

> *"In an efficient market, competition among the many intelligent participants leads to a situation where, at any point in time, actual prices of individual securities already reflect the effects of information based both on events that have already occurred and on events which as of now the market expects to take place in the future."*

However, especially in recent years, data analysts have developed various ways of estimating stock indices.

Contrary to Fama's hypothesis, according to analysts, if the prices are recently delivered, and the trends in movements are analysed correctly, it is possible to predict the future. Nevertheless, it should be kept in mind that stock movements are affected by many factors. Some of these factors are; policy, investor preferences, economy, politics, speculation, exchange rate, etc. [59, 66].

In addition to being affected by so many factors, time series have challenging problems such as non-linearity and non-stationarity, especially for financial market data [82, 89]. For instance, it has many upward and downward movements due to economic crises and speculations. To catch such a structure with a model or more than one model is a challenging real-world problem. First of all, analysts use classical statistical models such as autoregressive, moving averages, discriminating analyses, and correlations [49, 42] which are good at linear framework.

However, time series have both a linear and a non-linear part. Not only to model the non-linear part but also randomness and chaotic data of a time series, machine learning techniques are used [42, 85, 19]. The basic and the most used machine learning techniques in the literature are artificial neural networks (ANNs), support vector machines (SVMs), and random forests (RFs) [19].

It can be said that the neural networks and the tree-based algorithms are preferred compared to SVMs. According to the study of Labiad et al. [52], Gradient Boosted Trees and random forests are better than the SVMs [68]. Thus, the SVM algorithm is not covered in this thesis.

In gradient boosting algorithms, XGBoost distinguishes with its fast implementation, more accuracy, and preventing over-fitting therefore, it is preferred to Support Vector Regression (SVR), Random Forest (RF), and Gradient Boosting Decision Tree (GBDT) [23, 43].

For the neural network side, LSTM, a particular RNN version of Deep Learning, is a common choice to forecast financial time series. Since time series naturally includes a time-dependent variable, thereby LSTM intrinsically is pertinent [78].

According to a survey published in 2010, hybrid models are the second most preferred structures (the most preferred ones are ANNs) [13]. Hybrid models usually combine a linear and a non-linear model, i.e., ANNs and ARIMA [70]. However, in this thesis, two machine learning techniques are combined as a hybrid model unlike the ones used in literature because these models as universal models can capture not just the non-linear behaviour but also the linear one.

1.3 Structure of the Thesis

The structure of this thesis is as follows:

- In this Chapter, the thesis's motivation and the detailed literature review rele-

vant to the study are given.

- In Chapter 2, the required background information is exhaustively expounded, including a stock market index, time series, and machine learning algorithms.
- In Chapter 3, implementation is carried out so that the data and the algorithms are exemplified.
- In Chapter 4, the conclusion and the future work are emphasized.

CHAPTER 2

PRELIMINARIES

In this chapter, the essential concepts needed to use machine learning algorithms in time series analysis are covered. This chapter is believed to be significant to understand the details because the machine learning models are built on these bases. This chapter consists of five main parts; stock market index, log-return data, time series, residuals, and machine learning algorithms. Firstly, stock market indices and log-returns are defined. Secondly, the time series, one of the most popular tools used in stock market index prediction, and the residuals are described. Next, the machine learning algorithms are given with their mathematical background. It includes all necessary algorithms, their hyper-parameters, and error measures.

2.1 Stock Market Index

The statistical tool which indicates the changes in the stock market is called the stock market index. Indices are created in a few ways. In order to construct an index, a few similar kind of stocks are selected from the securities already listed on the exchange and then grouped to form a general sense about the economy of the corresponding country. There are variety of requirements for stock selection, but there are no unique requirements that are necessary. It is for this purpose that different stock indexes can be generated using different criteria. Thanks to stock market indices, the investors can understand the market and quickly compare the market's present and past values. Therefore, investing becomes meaningful and derives a profit for those who correctly analyse the market and make decisions in this direction.

2.2 Log-Return Data

Return is defined as

$$r_i = \frac{p_i - p_{i+1}}{p_{i-1}} \tag{2.1}$$

where r_i is the return and p_i is the price at time *i* for a security. Using returns instead of prices has a specific benefit: normalization. If there are more than two variables then normalization puts all variables in a same comparable metric. This step is essential for both multidimensional statistical and machine learning analysis. Moreover, using *log-returns* has major benefits in terms of both theoretical and algorithmic. First benefit is that if prices are distributed log normally then $\log(1 + r_i)$ is normally distributed;

$$\begin{aligned} 1 + r_i &= \frac{p_i}{p_{i-1}} \\ &= \exp^{\log(\frac{p_i}{p_{i-1}})}. \end{aligned}$$

Secondly, since returns are in general very small they are close in value to raw returns which is called *approximate raw-log equality*;

$$\log(1 + r) \approx r, \qquad r \ll 1. \tag{2.2}$$

The third benefit of log-return is the *time-additivity*. *Compounding return* which is calculated from an ordered sequence of *n* values is the running return over time. Corresponding sequence

$$(1 + r_1)(1 + r_2) \cdots (1 + r_n) = \prod_{i=1}^{n} (1 + r_i) \tag{2.3}$$

is unlikeable. Because probability theory tells the product of normally-distributed variables is not normal. Nevertheless, sum of normally-distributed uncorrelated variables is normal. So,

$$\log(1 + r_i) = \log(\frac{p_i}{p_{i-1}}) \tag{2.4}$$

$$= \log(p_i) - \log(p_{i-1}) \tag{2.5}$$

yields that compounding returns are normally distributed. If there are n variables then;

$$\begin{aligned} \sum_{i=1}^{n} \log(1 + r_i) &= \log(1 + r_i) + \cdots + \log(1 + r_n) \\ &= \log(p_n) - \log(p_0). \end{aligned}$$

In other words, difference between initial and final values is the compound return over *n* periods of time. Considering algorithmic complexity, Reducing $\mathcal{O}(n)$ multiplications to $\mathcal{O}(1)$ additions is a huge thing especially for large *n*. Besides, central limit theorem states that sample average of this sum will converge to normality.

Another benefit of log-return is its *mathematical ease*. It is known that the exponential function has the property that

$$\int e^x dx = e^x + c$$
$$\frac{d}{dx} e^x = e^x$$

where c is a constant of integration. This unique situation is beneficial, especially for financial mathematics, rested upon continuous-time stochastic processes.

Finally, the fifth benefit of log-return is its *numerical stability*. Since log-returns are small variables, the addition of them is safe. However, multiplication of them is not reliable. This serious problem should be solved by either modifying the algorithm or transforming it into numerically safe summation via logarithm.

2.3 Hybrid Models in Time Series

Time series might have both a linear and a non-linear parts with different features, so they should be modelled in different ways. To forecast time series data, this became an issue due to classical statistical methods that cannot capture the non-linear part of the time series. It only models the linear part. Since it is hard to know about data completely, the hybrid models which combine both linear and non-linear structures are preferable. To understand a time series completely, one should solve both parts: linear and non-linear. This problem led researchers to residual analysis, analysis of subtracting predicted values from real values. According to Zhang [90], neither applying only ARIMA nor only ANN helps to examine time-series fully, and it should be used together as a hybrid model. Time series consists of addition of two essential parts; linear L_t and non-linear N_t such that $y_t = L_t + N_t$. In Zhang's study, firstly, the ARIMA model is applied to examine the linear part as a statistical method. Then,

residuals only have a non-linear relationship to the model. Let $e_t = y_t - \hat{L}_t$ refers to residuals; where $\hat{L}_t$ is the forecast value at time t. If there is still a linear correlation in residuals, linear models cannot determine this situation. Even though residuals passed diagnostic tests, it might be incorrect because residuals cannot determine the non-linear structures. So, even if the model passed diagnostics tests, non-linear relations might not be modelled properly. In this situation, residual modelling with machine learning models helps discover the non-linear part of the time series. In Zhang's article, he used ARIMA for modelling the linear part and ANN for the non-linear part, and then he combined them to improve overall forecasting performance. Therefore, he concludes that to model a time series with different models as linear and non-linear parts separately could be more successful than using a single model.

Time series used to describe and analyse various series is a collection of data points based on time index. One of its aims is to forecast and to give good results for the future. Because of this nature of time series, people can make a strong decision about future for many areas. Financial market predictions, weather forecasts, and security applications are examples of where time series frequently used. From the mathematical point of view, it can be defined as a stochastic process

$$\{Y_t\}_{t=-\infty}^{\infty}$$

with a collection of random variables where t represents the time $t = 0, \pm 1, \pm 2, \ldots$.

Time series have three characteristic functions, the mean function, the auto-covariance function, and the autocorrelation function [87].

Definition 1. The mean function [25] is defined as

$$\mu_t = E[Y_t] \tag{2.6}$$

where $E[Y_t]$ is the expected value of the process at time t. The mean function exists if and only if $E \mid Y_t \mid < \infty$.

Definition 2. The autocovarince function [25] is defined as

$$\begin{aligned} \gamma_{t,s} &= Cov(Y_t, Y_s) \\ &= E[(Y_t - \mu_t)(Y_s - \mu_s)]. \end{aligned} \tag{2.7}$$

Definition 3. The autocorrelation function [25] is defined as

$$\begin{aligned} \rho_{t,s} &= Corr(Y_t, Y_s) \\ &= \frac{\gamma_{t,s}}{\sqrt{(\gamma_{t,t})(\gamma_{s,s})}}, \quad -1 \leqslant \rho_{t,s} \leqslant 1. \end{aligned} \tag{2.8}$$

- If $\rho(t, s) = +1$, then there is an exact positive linear association;
- If $\rho(t, s) = 0$, then there is no linear association at all;
- If $\rho(t, s) = -1$, then there is an exact negative linear association.

It should be considered that auto-covariance function (ACF) and partial ACF (PACF) concepts to decide the model. Therefore for a stationary process $\{Y_t\}$, the autocorrelation function between Y_t and Y_{t-k} is

$$\rho_k = Corr(Y_t, Y_{t-k}), \tag{2.9}$$

so the emphACF could be written as;

$$\begin{aligned} \rho_k &= Corr(Y_t, Y_{t-k}) \\ &= \frac{\gamma_k}{\gamma_0}. \end{aligned} \tag{2.10}$$

The Partial Autocorrelation Function shortly *PACF* is the correlation between Y_t and Y_{t-k} with removed linear dependency on the variables between these two terms.

To handle the stochastic process, there are some assumptions and the most important of these is *stationarity*. Basically, stationarity means that the statistical properties of the process do not change over time, i.e. it is time independent. There exists two types of stationarity: the strong and the weak stationarity. The weak stationarity is usually preferred because of the difficulty of verifying a distribution. To define a time series a weak stationary one, there are four properties [87]:

1. $E[Y_t] = \mu$,
2. $Var[Y_t] = \sigma^2$,
3. $Cov[Y_t, Y_{t-k}] = \gamma_k$,

4. $Corr[Y_t, Y_{t-k}] = \rho_k$

to satisfy for any time t.

2.3.1 Trend and Seasonality

If the process is not stationary, one can consider the concepts such as the trend, the seasonality, and thus making the process stationary. In a stationary time series, the mean function must be constant; otherwise, mean functions cause an upward or downward trend. The trend could be deterministic or stochastic. Depending on the type of trend, it could be removed with either de-trending or differencing through the process [87].

Furthermore, the series may have a seasonality effect. In such a case, there are some statistical tools necessary to understand the series, analyse it correctly, and make appropriate decisions. Models and forecasts made in line with these decisions will be much more accurate [87].

For this thesis in the prepreation setup, datasets are examined in terms of level, noise, seasonality and trend, but it is not found any outstanding results. Therefore, results does not contain an interpretation about decomposition of time series.

2.4 Residuals

Another important point in time series analysis is residuals. Residuals (r) are observed value (y) of the process at time t minus predicted value $\hat{y}$ of the process at time t: $r = y - \hat{y}$. If all residuals are close to 0, then it can be said that the model predicts perfectly. On the other hand, further from 0 residuals represents the poor predictions and the less accurate model. There are many reasons for using residuals. One reason is that information about the dataset properties, which are not visible at first glance, can be gathered from residuals. Another reason is that whether the chosen model is accurate can easily be shown by using residuals. Serial correlation or any non-linear structure in residuals shows that the fitted model cannot capture the

whole characteristics of the series. Hence, modelling residuals with another approach helps to capture the actual behaviour of the series.

2.5 Machine Learning Algorithms

In this section, machine learning algorithms, especially LSTMs and XGBoost, and their hybrid model are explained. Moreover, their hyper-parameters and error measures used in this thesis are given briefly.

Humans receive data from outside with their five sensory organs, and these data are transmitted to the brain, processed, and stored in a categorized form. Machines work similar. The main and the most crucial difference between a machine and a human is intelligence. Machines do not have creative ways to deal with data collected like humans. They may be expected to perform mechanical work quickly but not expected to gain experience or understand a theatre play. British mathematician Alan Turing first asked in 1955 [83] if machines could think, and this is indeed the beginning of the concept of artificial intelligence. Computers are machines that perform desired tasks and help people overcome their problems by following their instructions. Just as people perform the same task with various methods, machines implement different algorithms for the same problem. Accordingly, if different algorithms can work on a similar problem, the most crucial question is: which algorithm is the best. However, the answer to this question has not been given definitively today. Because we obtain a technology that develops every day, and we examine this technology on various data, with different methods [61]. Still, there are quite a few untested and unexplored elements. Although it is one step closer to achieving the best with each passing day, it cannot be stated that we are at that final point yet.

Machine Learning, part of Artificial Intelligence (AI), is a system that takes data and combines it with statistical tools then gives an output using different methods. In other words, machine learning systems sufficiently learn the complex structure of the series from its past values according to the given method and accurately predict the future data by properly implementing this structure when we have a long series. The main goal of learning is to establish a model that gives any data as input and gives a meaningful and desired output. Machine learning algorithms can be listed in three main titles: supervised learning, unsupervised learning, and reinforcement learning.

As is seen in Figure 2.1, even though there are quite a few machine learning methods, the important thing is to choose the right algorithm for the problem.

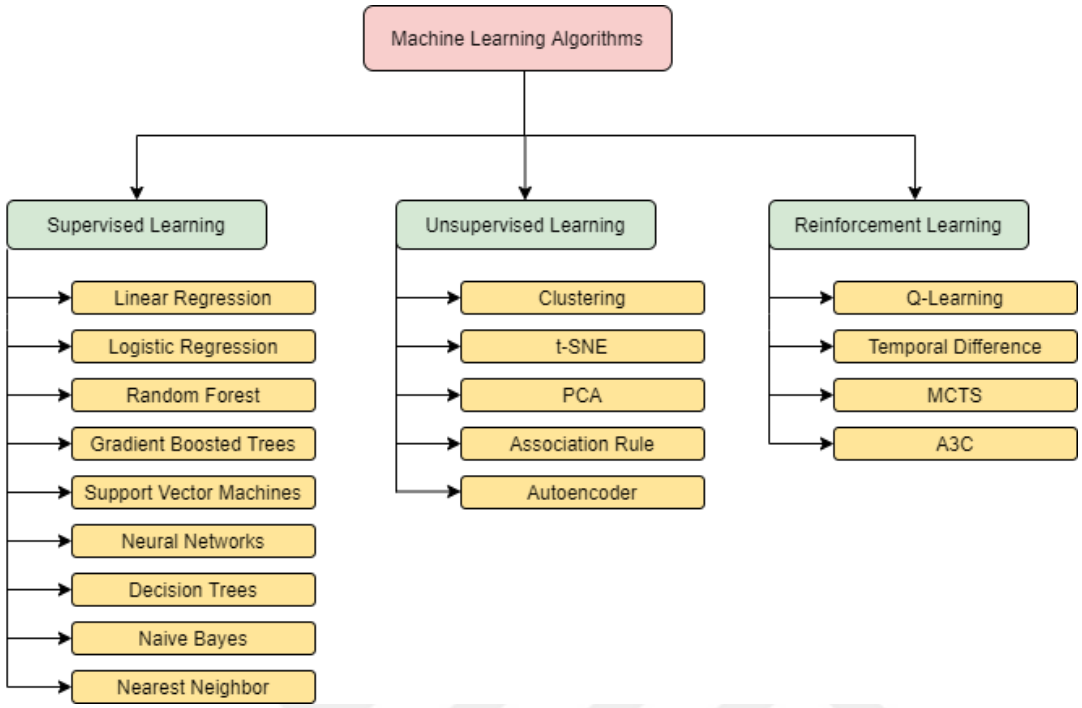


Figure 2.1: Some of the Machine Learning Algorithms

- **Supervised Learning**

  This type of learning is similar to the classroom teacher; it uses known and labelled data as input. That is, an input (x) is given as a guiding example, its corresponding output (Y) and a map connecting these two $(Y = f(x))$. The machine is expected to create a map from inputs to outputs, learn this rule and model it. Then, the model's accuracy is tested by giving another dataset that has never been seen [61]. Supervised Learning algorithms are divided into two categories, which can be shown in Figure 2.2;

  1. **Classification** is the method of obtaining or exploring a model or function which makes a difference in isolating the information into different categorical classes i.e., discrete values.
  2. **Regression** is the method of obtaining a model or function for recognizing the data into continuous genuine values rather than utilizing classes or discrete values.

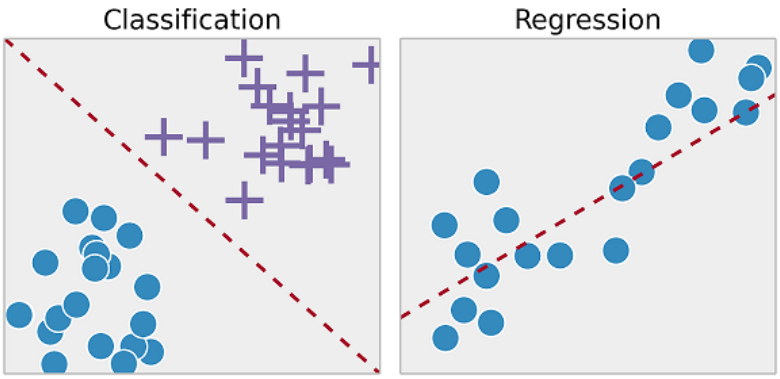


Figure 2.2: Classification(left) vs Regression(right) [47]

2.5.1 Neural Networks

Neural networks are a significant part of successful machine learning algorithms. Neural networks are developed to enable computers to think and to understand, like humans. Neural networks mimic the human brain. Just as neurons in a brain are connected by synapses, in machines, this is in the form of graph nodes being connected by weighted edges. The process of working the nervous system in humans represents the background of the thinking. This process is applied to computers, and a model is created using the neural network method. In a neural network, there are input and output neurons connected by weighted synapses. These weights control how much information will pass forward or backward through neural networks. Besides, weights can change in the forward or backward propagation parts. The iteration of forward and backward propagation for each data in the training set is called the learning process. The more extensive and more diverse the dataset set is, the more data it has been tested, and therefore the better the model will learn and perform well in forecasting [54]. An illustration of a neuron can be seen in Figure 2.3 as an example.

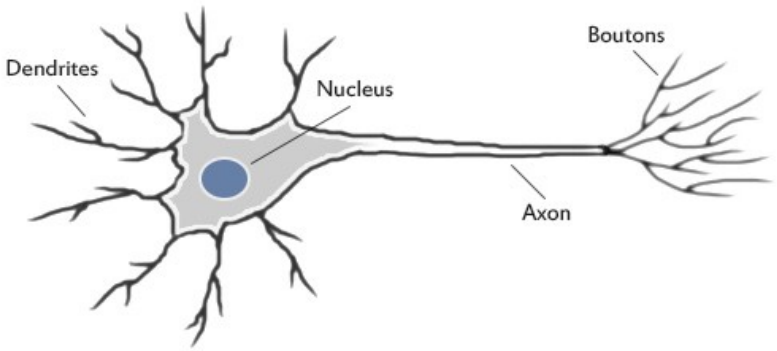


Figure 2.3: An illustration of a Neuron [75]

2.5.2 Artificial Neural Networks

Artificial neural networks are modelled inspired by biological neurons in humans. Accordingly, neurons in humans are activated in certain situations, and the activated neuron's output turns into an action. Artificial neural networks consist of interconnected layers of neurons and activation functions that enable these neurons to be activated or deactivated [76]. The ANN structure can be seen in Figure 2.4.

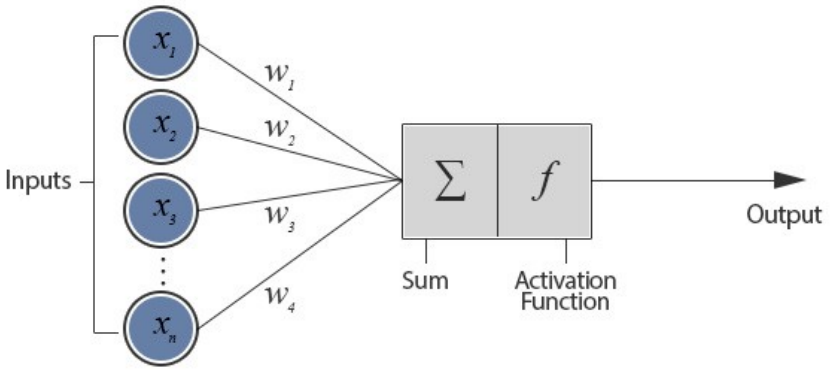
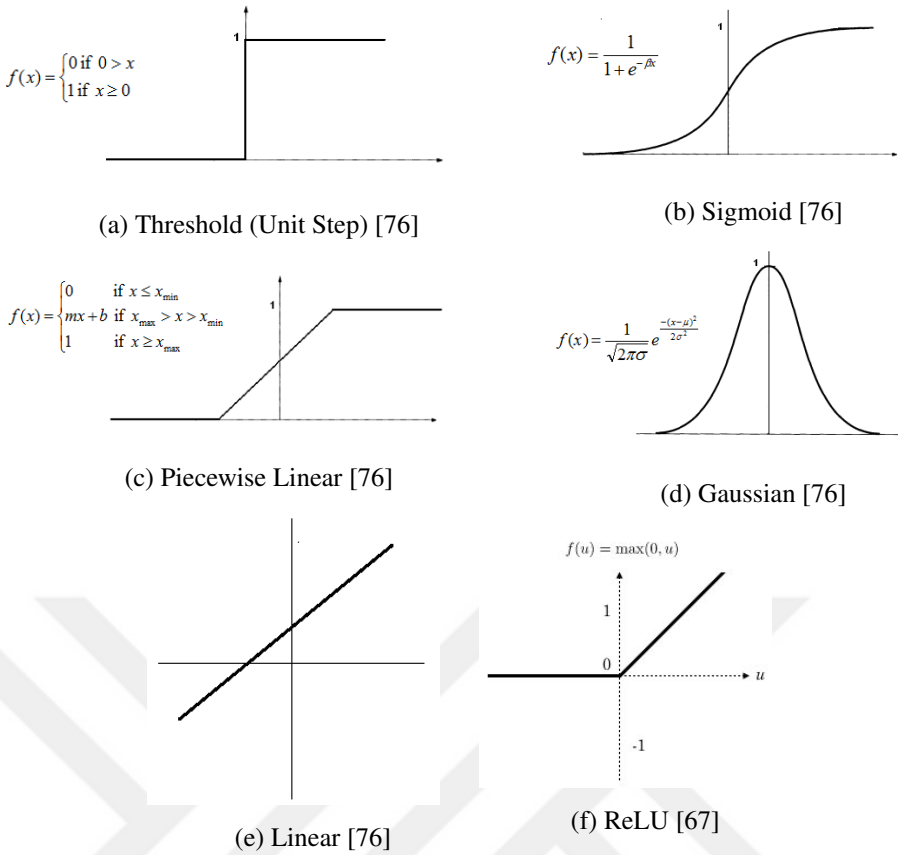


Figure 2.4: Artificial Neural Network [63]

ANN has three types of neurons, input nodes, hidden nodes, and output nodes. Input nodes are given information numerically. This information is presented with activation values, and each node is given a number; the one with the highest number performs the largest activation. After this stage, the information is transferred to the network. According to weights, the activation value passes from node to node. Each node collects the activation values it receives and then changes them according to the transfer function. This activation passes through the network, hidden layers, and reaches the output node in the final. According to the result, the difference between the predicted value and the observed value, namely error, is calculated. In this way, the weight is re-arranged according to the size of the error and is re-inserted into the system with back-propagation. The goal is to minimize the error [76]. Five common activation functions carry input signals to output signals, which are threshold, sigmoid, piecewise linear, linear, ReLU and Gaussian. The shapes of activation functions are in Figure 2.5.



(a) Threshold (Unit Step) [76]

(b) Sigmoid [76]

(c) Piecewise Linear [76]

(d) Gaussian [76]

(e) Linear [76]

(f) ReLU [67]

Figure 2.5: Activation Functions used in Artificial Neural Networks

Moreover, there are two types of ANN; Feedforward and Backward Neural Networks which can be seen in Figure 2.6.

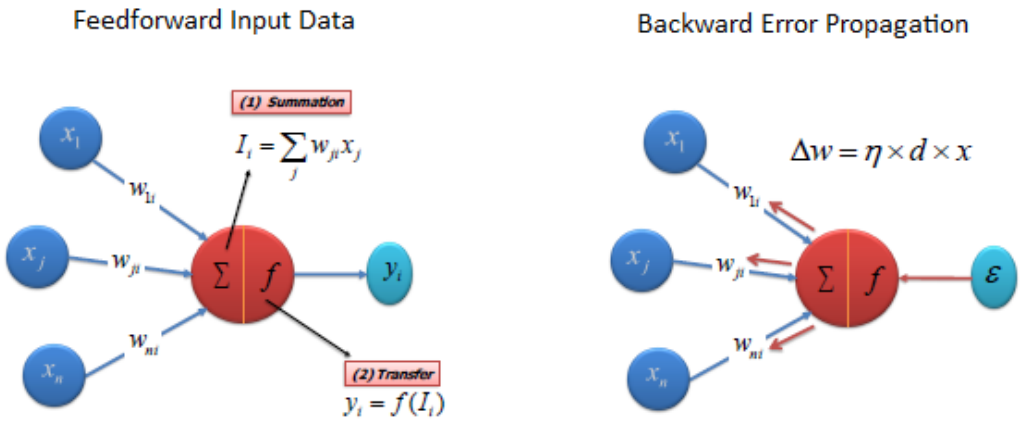


Figure 2.6: Feedforward (left) and Backward (right) Neural Networks [76]

2.5.3 Recurrent Neural Networks

A recurrent Neural Network is a generalized form of feedforward neural network system with internal memory. While it performs the same function for every input data, the output of the current input depends on the past computation. Once producing the output, it is sent back to the recurrent network. The other feedforward neural networks cannot manage their internal states to processing sequences of inputs, but RNN can. This feature makes RNN more applicable. Unlike the other neural networks, all the inputs are related to each other in RNN [4]. The structure of the Recurrent Neural Network can be seen in Figure 2.7.

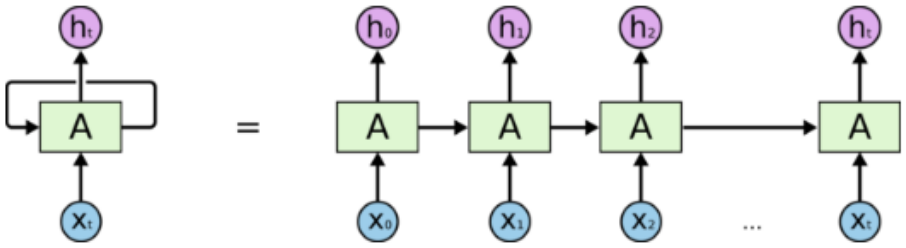


Figure 2.7: Recurrent Neural Network [62]

2.5.4 Long Short-Term Memory Units (LSTMs)

LSTM [39], which is one of the most popular algorithms, is a specifically developed version of Recurrent Neural Networks (RNN). Even though RNN's has been used in time series modelling well, vanishing gradient problem makes it hard to learn long term dependencies [6, 64, 65, 27, 72, 60, 37, 9, 58]; however LSTM solves this problem effectively by using a unique additive gradient structure [74]. The most important feature that distinguishes LSTM from others is that LSTM has a gating system. The gates are the forget gate, the input gate, and the output gate. Different gates are used to learn the network actively. In other words, the system finds out either which information should be forgotten and when, or which one should be updated [91].

Mathematically, LSTM structure can be expressed in the following equation;

$$
\begin{aligned}
f_t &= \sigma(W_f.[h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i.[h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_C.[h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o.[h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
\tag{2.11}
$$

where f_t is the forget gate layer at time t, h_{t-1} and x_t are the input variables, σ is a *sigmoid layer* which gives $n \in [0, 1]$ as an output. W_f is the weight of f, b_f is the bias vector of f and "." is the point-wise multiplication operator.

In the input gate layer, it has three parts which are i_t, $\tilde{C}_t$ and C_t. In i_t, it is the same as f_t, but i_t decides which information will be updated with σ, *sigmoid layer*. After that, $\tanh$ layer creates a vector including new informations between $[0, 1]$ as $\tilde{C}$. Then C_t, cell state, combines these steps and it continues by forgetting the information to be forgotten and saving the new information.

The last part is the output gate layer where σ decides which parts will be the output and $\tanh$ layer forces the cell state between $[0, 1]$ and multiplying with the output, a gate gives only the desired outputs.

The structure can also be seen in Figure Figure 2.8 [6] where $\bigotimes$ and $\bigoplus$ are the point-wise multiplication and the point-wise addition operators, respectively.

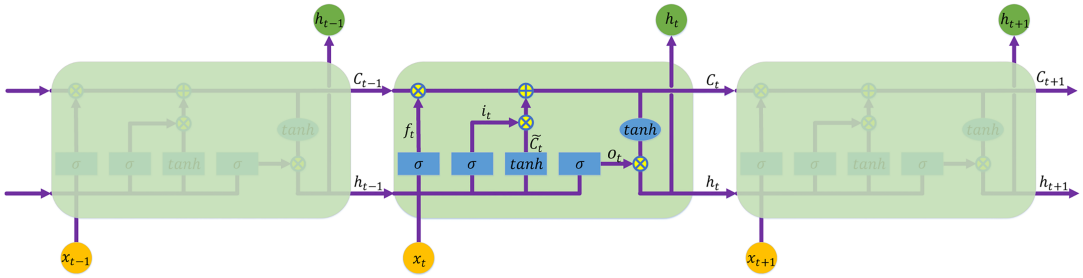


Figure 2.8: LSTM Algorithm [7]

2.5.5 Decision Trees

Decision tree is a technique used in data mining and data science to solve prediction problems in classification and regression model [86, 69, 5, 79, 38]. The components of a decision tree are the node representing a feature, the link (branch) representing a decision, and the leaf representing a result. The main goal is to minimize the error on each sheet for the entire dataset. If a decision tree is used for classification or regression, it is called a classification tree or a regression tree, respectively. The difference is that while regression trees deal with a continuous value, classification trees are for discrete values [73]. As with other supervised learning methods, part of the entire dataset is used to train the model. There is a rule for every branch in the decision tree, from nodes to leaves. For this reason, by looking at the decision tree diagram, it can be understood which of the factors used are effective in the decision variable and the relationship between the factors [69]. There are two steps for the decision tree diagram:

1. First of all, entropy values are calculated by Equation 2.12 for each factor.

$$E(S) = -\sum_{i=1}^{n} \frac{S_i}{S}(\log \frac{S_i}{S}), \tag{2.12}$$

   where S_i signifying the factors. The one that has the highest knowledge gain is obtained and selected as the starting node, i.e., root node. If entropy increases the uncertainty in the variables increases. So, when creating a tree structure, it starts from a low uncertainty and goes to a high one. The process continues until it reaches the decision variable (leaf). Information gain is calculated by subtracting the conditional entropy value from the total entropy value:

$$E(S_j|S_n) = \frac{S_i}{S}(\log \frac{S_i}{S}). \tag{2.13}$$

   So, the information gain is

$$E(S) - E(S_j|S_n).$$

   In this way, a tree diagram is created by determining the factor with the most significant knowledge gain and placing it in root nodes and subsequent nodes [10].

2. Secondly, by determining the minimum threshold for the number of observations per leaf, branches that are considered unimportant are pruned, and understandable pruning also increases the generalization power by removing the rules with few examples in the resulting decision tree [69]. Also, it eliminates over-fitting.

2.5.6 Bagging

The bagging, also known as bootstrap aggregation, is based on majority voting. The samples are bootstrapped every time the model is trained. When the samples are selected, they are used to train and validate the predictions. The samples are then put back into the training set. The samples are selected at random. This technique is known as bagging. There are pairs $(X_i, Y_i) \quad (i = 1, \ldots, n)$, where $X_i \in \mathbb{R}^d$ which denotes the d-dimensional predictor variable and the response $Y_i \in \mathbb{R}$ for regression or $Y_i \in \{0, 1, \ldots, J-1\}$ for classification with J classes. Moreover, the target function of interest is defined as $E[Y|X = x]$ for regression or for classification $\mathbb{P}[Y = j|X = x](j = 0, \ldots, J-1)$ as the multivariate function. The function estimator is

$$\hat{g}(\cdot) = h_n((X_1, Y_1), \ldots, (X_n, Y_n))(\cdot) : \mathbb{R}^d \to \mathbb{R}, \tag{2.14}$$

The bagging algorithm occurs in three steps:

1. Build a bootstrap sample $(X_1^*, Y_1^*), \ldots, (X_n^*, Y_n^*)$, randomly occurring n times with replacement from the data $(X_1, Y_1), \ldots, (X_n, Y_n)$.

2. Calculate the bootstrapped estimator $\hat{g}^*(\cdot)$ by the plug-in principle: $\hat{g}^*(\cdot) = h_n((X_1, Y_1), \ldots, (X_n, Y_n))(\cdot)$.

3. Repeat steps M times, where M is usually chosen 50 or 100, yielding $\hat{g}^{*k}(\cdot)$ for $k = 1, \ldots, M$. So, the bagged estimator is

$$\hat{g}_{Bag}(\cdot) = M^{-1} \sum_{k=1}^{M} \hat{g}^{*k}(\cdot). \tag{2.15}$$

Theoretically, as M tends to infinity

$$\hat{g}_{Bag}(\cdot) = \mathbb{E}^*[\hat{g}^*(\cdot). \tag{2.16}$$

In practice, the finite number M determines the Monte Carlo approximation's accuracy, but otherwise, it should not be viewed as a tuning parameter for the bagging. The empirical fact is that bagging improves regression and classification trees [15, 16, 12, 11, 14, 20] and reduces the variance of the model. An example of a bagging ensemble is Random Forest models.

2.5.7 Random Forest

Random Forest ensemble uses an outsized range of individual, unpruned decision trees that are created by randomizing the split at every node of the decision tree [26]. Every tree is probably going to be less correct than a tree created with precise splits. However, by combining many of those "approximate" trees in an ensemble, the accuracy can be improved doing higher than one tree with exact splits [73].

There are some procedures up to exploring the random forests. The common factor of these procedures is that for the kth tree, a random vector Θ_k is generated, independent of the past terms $\Theta_1, \ldots, \Theta_{l-1}$ but with the same distribution. A tree is constructed with the training set and Θ_k, resulting in a classifier $h(x, \Theta_k)$ where x is an input vector. For example, when bagging, a random vector Θ is generated as counts in N boxes due to N darts thrown at random into boxes, where N is the number of examples in the training set. If the sample is randomly split, Θ consists of the number of independent random integers from 1 to K. The nature and dimension of Θ depend on its use in constructing the tree. After a large N, the most popular class is selected. These whole procedures are called random forests [26].

Definition 4. A random forest is a set of classifiers with a tree structure $\{h(x, \Theta_k), k = 1, \ldots\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors. Each tree gives a single vote for the most popular class on the input x.

For both classification and regression, the random forest algorithm is similar [53]:

1. Draw n_{tree} bootstrap samples from the original data
2. For each of the bootstrap samples, develop an *unpruned* classification or regression tree, with the following modification: at each node, rather than choosing

the most proper distribution among all the predictors, randomly sample m_{try} the composition of the predictors and choose the most efficient distribution among these variables.

3. Predict new data by aggregating the predictions of the n_{tree} trees. For regression, it is averaging, and for the classification, it is the majority votes.

2.5.8 Boosting

Boosting algorithms have been proposed in the machine learning literature by Schapire [77] and Freund [29, 30]. Boosting is a sequential ensemble method that typically reduces the bias error and develops strong predictive models. The term 'Boosting' refers to a family of algorithms that convert a weak learner into a strong learner. The data samples are weighted, and therefore, some of the learners can take part in the recent sets more often. In each iteration, data points incorrectly predicted are identified, and their weight is increased so that the following learner focuses more attention on getting them correctly. The goal is to estimate a function $g : \mathbb{R}^d \to \mathbb{R}$, minimizing a loss

$$\mathbb{E}[\rho(Y, g(X))], \qquad \rho(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+, \tag{2.17}$$

based on data (X_i, Y_i) for $i = 1, \ldots, n$. This is the case both Y is continuous for regression problems and discrete for classification problems [11].

2.5.9 Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which generates a prediction model as a set of weak prediction models. It is based on the logic of minimizing the error by combining the next best model with the previous model. It performs this by setting target results for the next model. The target outcome depends on how much the change in the case estimate affects each case's overall estimate error. If a minor change in prediction results in a large change in error, then the next target result will be high. Nevertheless, if the minor change in the prediction does not make a difference in the error, then the next target result will

be zero; that is, it will not reduce the error. There is a system consists of a random output variable y and a set of random input variables $x = \{x_1, \ldots, x_n\}$ in the function estimation problem. Within a given training sample $\{y_i, x_i\}_1^N$, the main purpose is to find a function $F^*(x)$ mapping x to y.

$$F^*(x) = \underset{F(x)}{argmin} E_{y,x} \Psi(y, F(x)), \tag{2.18}$$

where $\Psi(y, F(x))$ is the loss function; with boosting, $F^*(x)$ approximates by an additive manner of the form

$$F(x) = \sum_{m=0}^{M} \beta_m h(x; a_m) \tag{2.19}$$

where $h(x; a)$ is a base learner and usually chosen to be a simple function. The coefficient $\{\beta_m\}_0^M$ and the parameter $\{a_m\}_0^M$ are fit to the training data. The initial guess starts with $F_0(x)$ and the process continues for $m = 1, 2, , \ldots, M$:

$$\begin{aligned} (\beta_m, a_m) &= \underset{\beta,a}{argmin} \sum_{i-1}^{N} \Psi(y_i, F_{m-1}(x_i) + \beta h(x_i; a)), \\ F_m(x) &= F_{m-1}(x) + \beta_m h(x; a_m). \end{aligned} \tag{2.20}$$

Gradient boosting algorithm [31] solves (2.20) approximately for an arbitrary loss functions in two steps. First, the function $h(x; a)$ is fit by least squares

$$a_m = \underset{a,\rho}{argmin} \sum_{i=1}^{N} [\tilde{y}_{im} - \rho h(x_i; a)]^2 \tag{2.21}$$

to the current residuals

$$\tilde{y}_{im} = - \left[\frac{\partial \Psi(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}, \tag{2.22}$$

then, the optimal value of the coefficient β_m is determined:

$$\beta_m = \underset{\beta}{argmin} \sum_{i=1}^{N} \Psi(y_i, F_{m-1}(x_i) + \beta h(x_i; a_m)). \tag{2.23}$$

This strategy replaces a potentially difficult function optimization problem with a least squares problem followed by a single parameter optimization based on the general loss criterion Ψ. Gradient tree boosting specializes this attitude in the case where the basic learner $h(x; a)$ is a regression tree of the terminal node L. At each iteration m, a regression tree partitions the space x into L-disjoint regions $\{R_{lm}\}_{l=1}^{L}$ and

predicts a distinct constant value in each one:

$$h(x; \{R_{lm}\}_1^L) = \sum_{l-1}^{L} \overline{y}_{lm} 1(x \in R_{lm}) \tag{2.24}$$

where

$$\overline{y}_{lm} = mean_{x_i \in R_{lm}}(\tilde{y}_{im}). \tag{2.25}$$

The parameters of this base learner are the splitting variables and the corresponding split points that define the tree, which in turn define the corresponding regions $\{R_{lm}\}_1^L$ of the partition at the mth iteration. These are induced in a "best-first" top-down manner using a least squares splitting criterion [33]. With regression trees, β_m can be solved separately within each region R_{lm} defined by the corresponding terminal node l of the mth tree. Since tree defined by (2.25) predicts a constant value $\overline{y}_{lm}$within each region R_{lm}, the solution of (2.24) reduces to a simple location estimate based on criterion Ψ:

$$\gamma_{lm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{lm}} \Psi(y_i, F_{m-1}(x_i) + \gamma). \tag{2.26}$$

Consequently, $F_{m-1}(x)$ is updated in the corresponding region

$$F_m(x) = F_{m-1}(x) + v \cdot \gamma_{lm} 1(x \in R_{lm}), \tag{2.27}$$

where the shrinkage parameter $0 < v \leq 1$ controls the learning rate. See Algorithm 1

Algorithm 1 Gradient Boosting Algorithm

$F_0(x) = argmin_\gamma \sum_{i-1}^{N} \Psi(y_i, \gamma)$

for m = 1 to M **do**:

$\tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, i = 1, N$

$\{R_{lm}\}_1^L = L -$ terminal node tree$(\{\tilde{y}_{im}, x_i\}_1^N)$

$\gamma_{lm} = argmin_\gamma \sum_{x_i \in R_{lm}} \Psi(y_i, F_{m-1}(x_i) + \gamma)$

$F_m(x) = F_{m-1}(x) + v \cdot \gamma_{lm} 1(x \in R_{lm})$

end **for**

2.5.10 eXtreme Gradient Boosting (XGBoost)

eXtreme Gradient Boosting, in short XGBoost, a gradient boosted tree based algorithm, is introduced by Tianqi Chen and Carlos Guestrin in 2016 [23]. It is a scalable

end-to-end tree boosting method, which has been widely used and achieved state-of-the-art classification and regression efficiency. XGBoost can carefully help tackle over-fitting, properly promote tree construction parallelization, and speed up execution [92, 55]. In order to define XGBoost deeply, there are necessary headlines [23] which are:

- Regularized Learning Objective
- Gradient Tree Boosting
- Shrinkage & Column Subsampling
- Split Finding Algorithms
  - Basic Exact Greedy Algorithm
  - Approximation Algorithm
  - Weighted Quantile Sketch
  - Sparsity-aware Split Finding

Regularized Learning Objective

The given dataset $\mathcal{D} = \{(x_i, y_i)\}$, where $|D| = n$, $x_i \in \mathbb{R}^m$, $y_i \in \mathbb{R}$, with n-examples and m-features uses K-additive functions to predict the output:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i) \tag{2.28}$$

where

$$f_k \in \mathcal{F} = \{f(x) = w_{q(x)}\}$$

is an independent tree structure $q : \mathbb{R}^m \to T$ where $\mathcal{F}$ is the space of regression trees (CART), T is the number of leaves in the tree, $w \in \mathbb{R}^T$ is the weight of the leaf. Therefore, the aim is to minimize the regularized objective;

$$\mathcal{L}(\theta) = \sum_i \ell(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{2.29}$$

where

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

is the complexity of the model, ℓ is the differential convex loss function which measures the difference between the prediction ($\hat{y}_i$) and the target (y_i) value. It is noted that an extra regularization term prevents over-fitting [23].

Gradient Tree Boosting

It is impossible to optimize Equation (2.29) with practical approach in Euclidean space. Therefore, the additive manner is considered in the model to be trained. Normally, $\hat{y}_i^{(t)}$ is the prediction of the i-th instance at the t-th iteration, yet to minimize the objective f_t is added [23].

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

After second-order approximation, $\mathcal{L}^{(t)}$ becomes

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [\ell(yi, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_,)] + \Omega(f_t)$$

, where

$$g_i = \partial_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}^{(t-1)}),$$
$$h_i = \partial^2_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}^{(t-1)}).$$

To simplify we remove the constant terms, so the objective function at step t becomes

$$\tilde{\mathcal{L}}^t = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t). \tag{2.30}$$

Letting $I_j = \{i \mid q(x_i) = j\}$ as the instance set of leaf j, (2.30) can be re-written as

$$\begin{aligned} \tilde{\mathcal{L}}^t &= \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2} \lambda \sum_{j_1}^{T} w_j^2 \\ &= \sum_{j_1}^{T} [w_j \sum_{i \in I_j} gi + \frac{1}{2} w_j^2 \sum_{i \in I_j} h_i + \lambda] + \gamma T. \end{aligned} \tag{2.31}$$

The computed optimal weight w_j^* of leaf j is therfore

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{2.32}$$

and the corresponding optimal value which can be used for measuring the quality of a tree structure q is

$$\tilde{\mathcal{L}}^t(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T. \tag{2.33}$$

Shrinkage & Column Subsampling

In addition to the Regularized Learning Objective and the Gradient Tree Boosting techniques, Shrinkage, and Column (Feature) subsampling can also be used to prevent over-fitting. The shrinkage technique, which is introduced by Friedman [32] is the first technique. After every boosting step, recently included weights are scaled by shrinkage, decreasing the impact of each tree and leaves according to factor μ. The second technique, column (feature) subsampling, which is used in Random Forest [17, 34] is inscribed that column subsampling avoids over-fitting in comparison with the traditional way [23].

Split Finding Algorithms

There are some important problems in tree learning, but one of the key problems is to find the best split. To do so, a split finding algorithm enumerates over all possible splits on all available features. If I_L and I_R are accepted as the instance sets of the left node and the right node respectively, then letting $I = I_L \cup I_R$ allows one to evaluate the split candidates by

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma. \tag{2.34}$$

Basic Exact Greedy Algorithm

The important thing is to find the best split by using (2.34), so a split finding algorithm tries all possible splits on all columns, which is called *Exact Greedy Algorithm*. In order to do this effectively, the algorithm first sorts the data and then review data in that order [23].

Approximate Algorithm

No matter how good the basic exact greedy algorithm is, there are two setting issues. Since Basic Exact Greedy Algorithm cannot overcome all possible splitting points and distribute, an Approximate Algorithm is needed for these two settings. In short, the algorithm asserts the applicant splitting points at first. Then, it determines the continuous features. Lastly, the algorithm attains the relevant statistics and finds the best solution according to relevant statistics.

There are two types of algorithms, such that users are free to decide. These two types are the global variant and the local variant. The global variant lists all the applicant splits at the beginning of the tree construction and use this at all levels. However, the local variant re-sets after each split [23].

Weighted Quantile Sketch

Proposing candidate split points is crucial for the Approximate Algorithm. Let, $\mathcal{D}_k = \{(x_{1k}, h_1), (x_{2k}, h_2)...(x_{nk}, h_n)\}$ is the k-th feature values and second order gradient statistics of each training samples. Rank function $r_k : \mathbb{R} \to [0, +\infty)$ describes the quotient of samples whose feature value k is smaller than z; and this function can be defined as

$$r_k(z) = \frac{1}{\sum_{(x,h)\in\mathcal{D}_k} h} \sum_{(x,h)\in\mathcal{D}_k, x<z} h. \tag{2.35}$$

The aim is to find applicant split points $\{s_{k1}, s_{k2}, ..., s_{kl}\}$ such that

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon, \qquad s_{k1} = \min_i x_{ik}, \quad s_{kl} = \max_i x_{ik}, \tag{2.36}$$

where ϵ is an approximation factor. Also weighted squared loss is

$$\sum_{i=1}^{n} \frac{1}{2} h_i (f_i(x_i) - g_i/h_i)^2 + \Omega(f_t) + constant,$$

where labels are g_i/h_i and h_i represents weights. When the dataset is large, it is hard to find candidate splits that satisfy the criteria. For this reason, it is introduced a new distributed weighted quantile sketch algorithm which data structure supports *merge* and *prune* operations [23].

Sparsity-aware Split Finding

It is very prevalent in real-world problems that the input variable is sparse, and there are many reasons for sparsity such that missing values, frequent zero entries. Even so, it is necessary to make data-aware of the sparsity. A new model is introduced to create sparsity-aware data, so if there is a missing value in data, then the sample is classified in direction by default. There are two ways in each point, and the data itself learns which way is the best. Therefore, missing values are not a problem for XGBoost Algorithm [23].

2.5.11 Hybrid Models

Hybrid models are usually preferred for time series prediction. Since mostly time series has both linear and non-linear parts, the non-linear part may not be captured with linear statistical methods; therefore, hybrid models that combine the statistical methods with the machine learning methods are introduced [88, 35]. In detail, the linear part is modelled with the statistical method, and its residuals, which are the non-linear part of the series, are modelled with machine learning algorithms. Alternatively, in this study, two machine learning algorithms, LSTM and XGBoost, are run to predict stock market index time series. Similarly, each series is modelled with LSTM and XGBoost, separately. After that, residuals are obtained from each method. Then, the residuals are modelled with the another method, i.e., residuals of LSTM are modelled with XGBoost, and the residuals of XGboost are modelled with LSTM. With this structure, higher performance and accurate results are aimed.

2.5.12 Boruta Feature Selection Algorithm

Boruta is a feature selection algorithm that helps decide which of the variables in the dataset are important and which are not [51]. Feature selection is commonly a crucial step in machine learning algorithms. Datasets are typically represented with way too several variables for sensible model building. Typically most of those variables are irrelevant to the classification and regression, and clearly, their relevancy is not known in advance. Briefly, Boruta is predicated on an identical plan that forms the

random forest classifier's inspiration, namely, that by adding randomness to the system and collecting results from the set of randomized samples, one will decrease the misleading effect of random fluctuations and correlations. Boruta algorithm [50] can be expressed in nine steps:

1. Expand the information system by adding all variables to at least five attributes;
2. Remove correlations mix the added features;
3. Run a random forest classifier and get the Z scores;
4. Find the maximum Z score among shadow attributes (MZSA) and then specify every attribute that scored better than MZSA;
5. Do a two-sided test of equality with the MZSA for each undetermined importance attribute;
6. Consider the attributes which have importance significantly lower than MZSA as unimportant features and remove them from the system;
7. Consider the attributes which have importance significantly higher than MZSA as important features;
8. Remove all shadow attributes
9. Repeat the procedure until importance is identified for all available attributes.

2.5.13 Tuning Hyper-parameters

Tuning hyper-parameters, which are the key of the algorithms, make the structure more valid and make predictions more accurate. Consequently, tuning hyper-parameters are not unique. Depending on both the algorithm and the data, hyper-parameters could change for the ultimate goal. Moreover, each algorithm has its own hyper-parameters, so this subsection comprises two parts; LSTM Tuning Hyper-parameters and XGBoost Tuning Hyper-parameters.

2.5.13.1 Tuning Hyper-parameters of LSTM

- **lag input:** LSTM structure should capture time dependency in time series. Unfortunately, only the one-time step input does not make sense for the algorithm. Therefore, either creating an embedding matrix or manually lagging the data and providing a matrix helps capture time dependency. The number of lag input depends on the frequency of the data. For instance, if there is daily data, then the number of lag input could be 7, or if the data is monthly, then it could be 12.

- **epochs:** While the model is being trained, not all of the data participate in training simultaneously. They take part in training in a certain number of parts. The first part is trained, the model's performance is tested, and the weights are updated with backpropagation based on success. Then, with the new training set, the model is retrained, and the weights are updated again. This process is repeated in each training step to try to calculate the most suitable values of the weights for the model. Each of these training steps is called an epoch.

  Since the most appropriate weight values to solve the problem in deep learning is calculated step by step, success will be low in the first epochs, but it increases afterwards. However, after a particular step, the learning status of the model will decrease significantly.

- **batch size:** The batch is the number of values used in every epoch to train the model. If it is too high, then the model takes too long to achieve convergence; however, if the batch size is too small, it will obstruct the model from achieving accurate performance [44].

- **units in hidden layer:** The number of hidden units is a direct representation of the learning capacity of a neural network. Even though, similar to other hyper-parameters, there is no best number for all data. It could be selected either arbitrarily or empirically; still, it could be rearranged according to the state of affairs. In case of necessity, the value could be changed, and re-run the model to see the affects in the training accuracy. Using more units makes it more potentially to correctly memorize the complete training set, although it will take a longer time and it will have the risk of over-fitting.

- **optimizer:** One of the most critical hyper-parameters is an optimizer, which decides the learning rate. Several optimizers, such as Stochastic Gradient Descent (SGD), RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam, and Ftrl exist. The most common optimizers for time series prediction are SGD, RMSprop, and Adam [22]. However, Adam [46] is the first optimizer that comes to mind when there are large datasets and high dimensional parameters. Additionally, bias correction can be done with Adam [71].

- **loss:** The goal is to minimize the loss functions. There are three types of loss functions. The first one is regression loss functions, including squared error loss, absolute error loss, and Huber loss. Secondly, binary classification loss functions which are binary cross-entropy and Hinge loss. The last one is multi-class classification loss functions that have multi-class cross-entropy loss and Kullback Leibler Divergence loss. Since this study is considered regression, *mean squared error* is chosen as a loss parameter.

- **validation split:** This is used to control the model. The accuracy of the model could be destroyed whether the model is either under-fit or over-fit. The model is said to overfit when the model memorizes within the given dataset and cannot perform well on the test set. Otherwise, the model is said to under-fit when the model cannot learn with the train set but perform relatively well on the test set. Also, to tune hyper-parameters, it is necessary to control validation.

- **layer_dropout:** It is also used to avoid over-fitting. When the dropout hyper-parameter is tuned, it drops out the unnecessary neurons. For instance, if $layer\_dropout = 0.2$, it means that model drops out $\%20$ of all the neurons [1].

- **L2 Regularizer:** It is also known as *Ridge* regularization which preserves over-fitting. In the cost function

$$L(x, y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2, \tag{2.37}$$

the term

$$\lambda \sum_{i=1}^{n} \theta_i^2 \tag{2.38}$$

is the L2 Regularization. Ridge regression adds "squared magnitude" of coefficient as penalty term to the loss function.

2.5.13.2 Tuning Hyper-parameters of XGBoost

There are many XGBoost hyper-parameters, but only a few hyper-parameters used in this study are described in this section. For all hyper-parameters, the documentation of XGBoost could be viewed [2]. In Table 2.1 hyper-parameters with their default values that are used in this study can be seen. Additionally, three types of the booster to use are $dart, gbtree, gblinear$. While $dart$ and $gbtree$ use tree-based algorithms, $gblinear$ uses linear model. In this study, the tree-based algorithm $gbtree$ is preferred.

Table 2.1: Hyper-parameters of XGBoost

Hyper-parameter type	Hyper-parameter	Default Value & Range
Booster Hyper-parameters	max_depth	default = 6, range =[0,∞)
	$colsample\_bytree$	default = 1
	min_child_weight	default = 1, range =[0,∞)
	eta	default = 0.3, range =[0,1]
	$gamma$	default = 0, range =[0,∞)
	$subsample$	default = 1, range = (0,1]
Command line hyper-parameters	$nrounds$	-

- **max_depth:** It arranges the maximum depth of a tree. Increasing this hyper-parameter makes the model more complex and more likely to over-fit. Moreover, when max_depth gets larger, training time increases. It is used to control over-fitting issue.

- **colsample _ bytree:** It specifies fractions of features to choose randomly samples for each tree.

- **min_child_weight:** It identifies what number of examples min_child_weight nodes can represent to or their entropy as far as Hessians, an essentially significant hyper-parameter in preventing over-fitting and reducing the unpredictability of the model. The assurance of how far the nodes will part additionally influences the profundity of the tree. Utilizing cross-validation the fitting min_child_weight can be chosen for the model.

- **eta:** It stands for learning rate and makes the model more strong by shrinking the weights on each step.

- **gamma:** It is a L_0 regularization parameter to do a split; it specifies the minimum loss reduction. Increasing gamma makes the model more conservative.
- **subsample:** subsample represents the size of the example rate to utilize when preparing indicators. Its default esteem is 1, so there is no sampling, and all information is utilized. It is a boundary that helps prevent over-fitting.

2.5.14 Error Measures-Evaluation Criteria

Even though there is not a unique criterion for error measure, there are few common methods such as the *root mean squared error (RMSE)* [8], the *mean absolute percentage error (MAPE)* which were used in the M-competition as leading measure [56] and the *mean absolute scaled error (MASE)* [41]. To evaluate the performances of the different models on the different datasets, one error measure is not enough.

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(y_k - \hat{y}_k)^2} \tag{2.39}$$

$$MAPE = \frac{1}{N}\sum_{k=1}^{N}\left|\frac{y_k - \hat{y}_k}{y_k}\right| \times 100 \tag{2.40}$$

$$MASE = \text{mean}\left(\left|\frac{e_k}{\frac{1}{N-1}\sum_{k=2}^{N}|y_k - y_{k-1}|}\right|\right) \tag{2.41}$$

where the forecast error $e_k = y_k - \hat{y}_k$, y_k is the observed value and $\hat{y}_k$ is the forecasts at time k.

Since MASE is scale-free, both analyses can easily be used involving a single dataset and analyses containing multiple datasets. Also, MAPE is scale-free so that it can be used for comparing different datasets, but it may give infinite or undefined values [81]. According to Rob Hyndman, when these three measures were considered, MASE is the best error measure because of allowing to apply series on different scales, including negative or close to zero data [41].

CHAPTER 3

IMPLEMENTATION

In this chapter, analyses of the time series are presented in the light of information explained in Chapter 2 were included. At first, information of data is given. Then, the usage of algorithms is introduced. The comparison of the algorithms with each data, also corresponding tables and figures were explained in the sequel.

In this study, nine financial time series data as log-returns of stock market indexes and two machine learning algorithms and their combinations; in total, four different models are used.

All tests are carried out in R Programming Language, a freely available language for statistical computing and graphics. Additionally, Excel is used to calculate each data's log-returns, import the data from Excel as *cvs* file to R, and plot time series of both the original data and the log return data. Moreover, descriptive statistics are calculated in EViews. Finally, boxen & violin plots are created in Python.

3.1 Data Description

For this study, nine different stock market time series data are downloaded from Yahoo Finance website, and The Wall Street Journal web page, which is listed in Table 3.1 with their names, notations, and currencies in the market.

Stock market indices have been carefully selected considering their market orientation, size, and corresponding country development. Also, priority has been given to the stock market indices such as European, Asian, and U.S. markets that are the most

Table 3.1: Details of Datasets

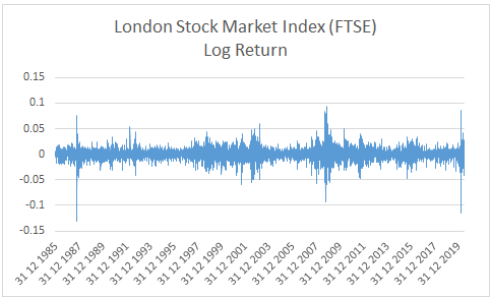
Data	**Notation**	**Currency**
London Stock Market Index	Financial Times Stock Exchange (FTSE 100)	GBP
German Stock Market Index	Deutscher Aktienindex (DAX)	USD
French Stock Market Index	CAC 40 (FCHI)	EUR
Hong Kong Stock Market Index	Hang Seng Indexes (HSI)	HKD
Japan Stock Market Index	Nikkei 225 (N225)	JPY
China Stock Market Index	Shanghai SE (SSE)	CNY
New York Stock Market Index	NYSE Index (NYA)	USD
U.S. Stock Market Index	S&P Index 500 (GSPC)	USD
U.S Stock Market Index	NASDAQ (IXIC)	USD

used in analyses and articles [6]. Therefore, there are three European markets such that London, Germany, and France; three Asia markets such that Hong Kong, Japan, and China; and three U.S. markets such that New York, S&P 500, and NASDAQ.

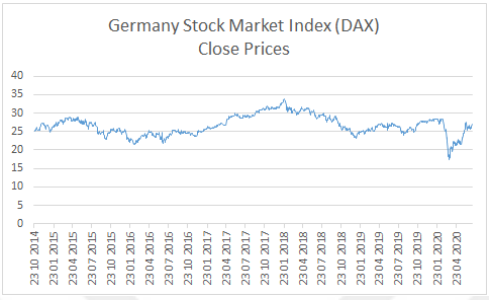
All data are in daily frequency; however, since it is a Stock Market Index, no value has been recorded when the Stock Exchange is closed, i.e., on weekends and national holidays. Also, the days of the stock market are closed for all countries are not the same, but this study aims to compare the machine learning algorithms' performances, so no action has been taken on this issue. The only consideration is the size of the dataset.

Line plots of each close prices data and log-returns of close prices' line plots can be seen in Figure 3.1, 3.2, and 3.3. When the data are examined separately, especially on New York, S&P 500 and NASDAQ, they all have increasing trends after years 1997-1998 which is quite essential problem for machine learning analysis because XGBoost algorithm is not as successful as LSTM for capturing the time series with the trend without scaling since it is a tree-based algorithm. Working with the log-return overcomes this problem since creating return makes the series stationary, so the algorithm's learning capacity may increase. Also, taking log-returns makes all data stationary, but some have variation a lot, especially at some points.

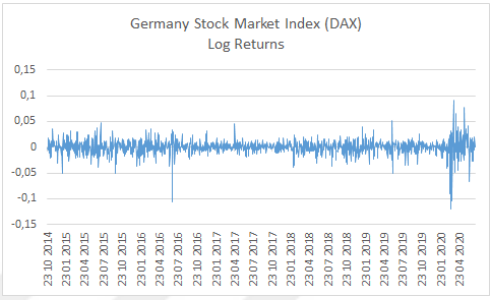
(a) Close Prices of London



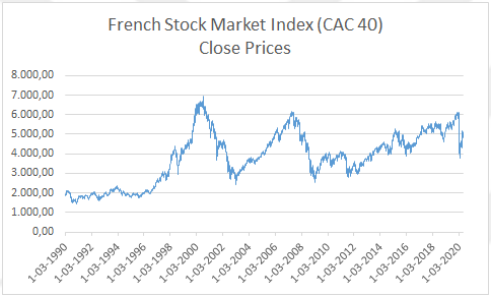
(b) Log-Return of London



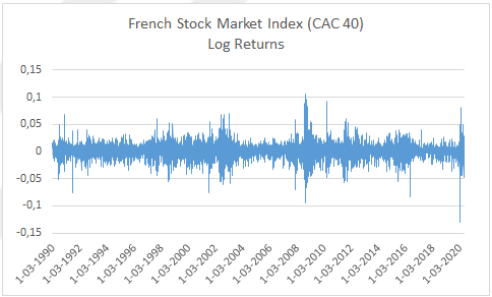
(c) Close Prices of Germany



(d) Log-Return of Germany

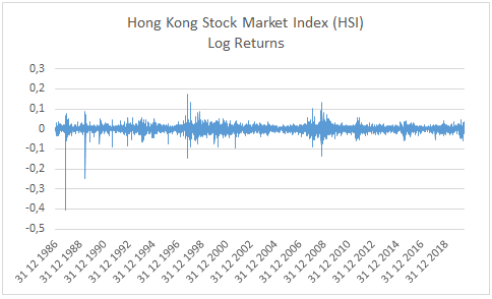


(e) Close Prices of France



(f) Log-Return of France

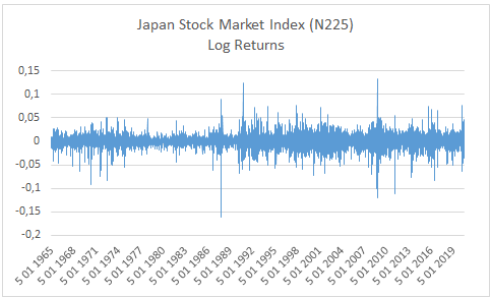
(g) Close Prices of Hong Kong



(h) Log-Return of Hong Kong

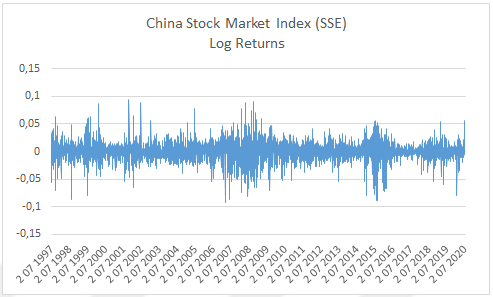
Figure 3.1: Line Plots of Close Price Data (left) and Log-Return Data (right)-1

(a) Close Prices of Japan

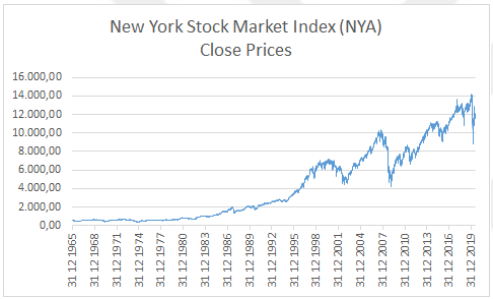


(b) Log-Return of Japan

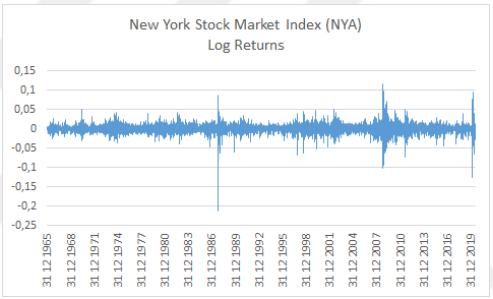
(c) Close Prices of China



(d) Log-Return of China

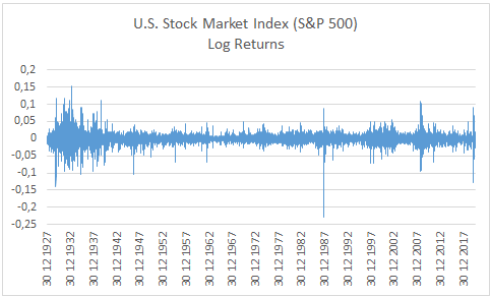


(e) Close Prices of New York



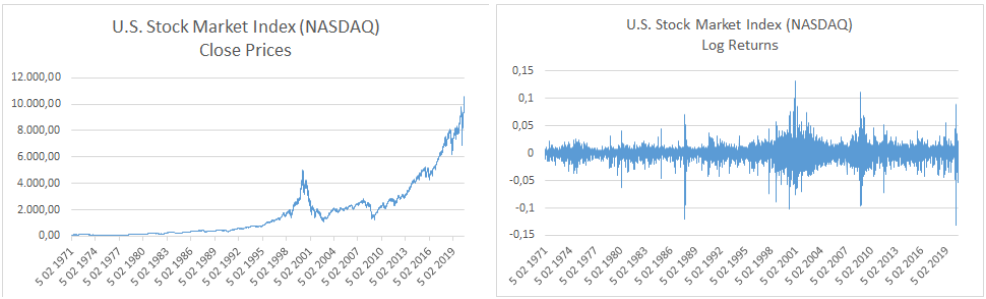
(f) Log-Return of New York

(g) Close Prices of S&P 500



(h) Log-Return of S&P 500

Figure 3.2: Line Plots of Close Price Data (left) and Log-Return Data (right)-2



(a) Close Price of NASDAQ (b) Log-Return of NASDAQ

Figure 3.3: Line Plots of Close Price Data (left) and Log-Return Data (right)-3

3.1.1 Descriptive Statistics of Data

In this section, close prices of Stock Market Indexes and log-returns of data are examined statistically. In other words, the violin plots, boxen plots and the descriptive statistics can be seen in Figure 3.4, Figure 3.5, Figure 3.6 and Table 3.2Table 3.3, respectively. All plots are generated in Python and the descriptive statistics are calculated in EViews. In the table of descriptive statistics, the mean, the median, the maximum and the minimum, the standard deviation, the skewness and the kurtosis, the Jarque-Bera and the probability values are calculated. The skewness value which measures the symmetry should be zero if the series normally distributed. Otherwise, the greater than zero values defines the right, and the smaller than zero values represents the left skewness. For the normally distributed series the kurtosis value which describes heavy-tailed or light-tailed should be three.

According to Table 3.2 and Table 3.3, dates are differ from each other so that lengths of observations change. For instance, Germany Stock Market Index has the minimum observations, on the other hand, S&P 500 Stock Market Index has the maximum observations, and the rest of the datasets are between these two extreme ones.

In Table 3.2, there is no a dataset whose skewness value is zero and kurtosis value is three, but, some of the dataset are close to being normally distributed. The skewness of Germany, France and Hong Kong Stock Market Indexes are close to zero. However, rest of the datasets' skewness are greater than zero which means positive skewness in other words the right-handed tail is larger than the left-handed tail. Moreover, it can be said that kurtosis value of Germany and Japan Stock Market Indexes

are close to three. However, except China and NASDAQ, kurtosis values are smaller than three so that they have lighter tails than a normal distribution, but China and NASDAQ have heavy tails.

In Table 3.3, descriptive statistics of log-return data, similar to close prices, there is no such a dataset whose skewness is zero and kurtosis is three. But, the minimum skewness is recorded in France with 0.21. Others are either greater or smaller than zero. While, London and Germany datasets have negative skewness in other words left skewness, all Asia and U.S. markets have right skewness. Also, values of kurtosis are quite bigger than three, so all datasets have heavy tails. According to results of descriptive statistics, there is neither similarities nor differences between markets in terms of geographical; European, Asia and U.S..

Table 3.2: Descriptive Statistics of Close Prices Data

	London	Germany	France	Hong Kong	Japan	China	New York	S&P 500	NASDAQ
Sample	12/31/1985 7/06/2020	10/23/2014 7/06/2020	3/01/1990 7/06/2020	12/31/1986 7/06/2020	1/05/1965 7/07/2020	7/02/1997 7/06/2020	12/31/1965 7/06/2020	12/30/1927 7/10/2020	2/05/1971 7/10/2020
Observations	8820	1434	7696	8270	13654	5741	13720	23241	12467
Mean	4812.171	26.68202	3817.422	14884.84	12631.55	2318.207	4289.698	476.7728	1798.357
Median	5235.490	26.43500	3910.585	13891.65	11541.55	2181.941	2593.895	99.29000	1026.410
Maximum	7877.450	33.80000	6922.330	33154.12	38915.87	6092.057	2593.895	3386.150	10617.44
Minimum	1370.100	17.38500	1441.000	1894.900	1020.490	1011.499	347.7700	4.400000	54.87000
Std. Dev.	1780.380	2.642148	1311.927	7993.754	7896.409	908.2576	347.7700	707.1302	2077.470
Skewness	0.328543	0.075161	0.072397	0.080000	0.508235	0.867011	0.698592	1.784592	1.631817
Kurtosis	1.844208	2.938572	2.007477	1.904673	2.810221	3.948960	2.180861	1.784592	5.280342
Jarque-Bera	649.5993	1.575610	322.6120	422.2325	608.3009	934.6721	1499.545	18711.63	8234.074
Probability	0.000000	0.454842	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

Table 3.3: Descriptive Statistics of Log-Return Data

	London	Germany	France	Hong Kong	Japan	China	New York	S&P 500	NASDAQ
Sample	12/31/1985 7/06/2020	10/23/2014 7/06/2020	3/01/1990 7/06/2020	12/31/1986 7/06/2020	1/05/1965 7/07/2020	7/02/1997 7/06/2020	12/31/1965 7/06/2020	12/30/1927 7/10/2020	2/05/1971 7/10/2020
Observations	8819	1433	7695	8269	13653	5740	13719	23240	12466
Mean	0.000169	4.62e-05	0.000133	0.000282	0.000212	0.000178	0.000229	0.000224	0.000374
Median	0.000397	0.000147	0.000377	0.000612	0.000396	0.000283	0.000500	0.000477	0.001067
Maximum	0.093843	0.092150	0.105946	0.172470	0.132346	0.094010	0.115258	0.153661	0.132546
Minimum	-0.130286	-0.120154	0.130983	0.405420	0.161354	0.092561	0.212863	0.228997	0.131492
Std. Dev.	0.011111	0.014117	0.013819	0.016478	0.012765	0.015442	0.010306	0.012023	0.012519
Skewness	-0.588515	-1.102095	0.212471	2.304205	0.410531	0.353807	1.102782	0.481599	0.380913
Kurtosis	13.64261	15.25181	8.740516	60.69932	12.41006	8.108591	28.51225	22.15637	13.52677
Jarque-Bera	42157.46	9252.727	10623.62	1154369.	50756.93	6361.455	374837.1	356244.3	57859.57
Probability	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

According to Figure 3.4, close prices of Hong Kong and Japan Stock Market Indexes look like to have some outlier points. Also, maximum values of nine datasets are different even though minimum values look like the similar. It can be said that all datasets except Germany have some outliers. Again, except Germany, France and London Stock Market Indexes are similar to each other. Moreover, New York, S&P 500 and NASDAQ Stock Market Indexes have similar structure. Hong Kong and

Japan Stock Market Indexes are similar, but China has a different structure. As a result, this outputs are not enough to conclude having either similarities or differences based on geographical features.

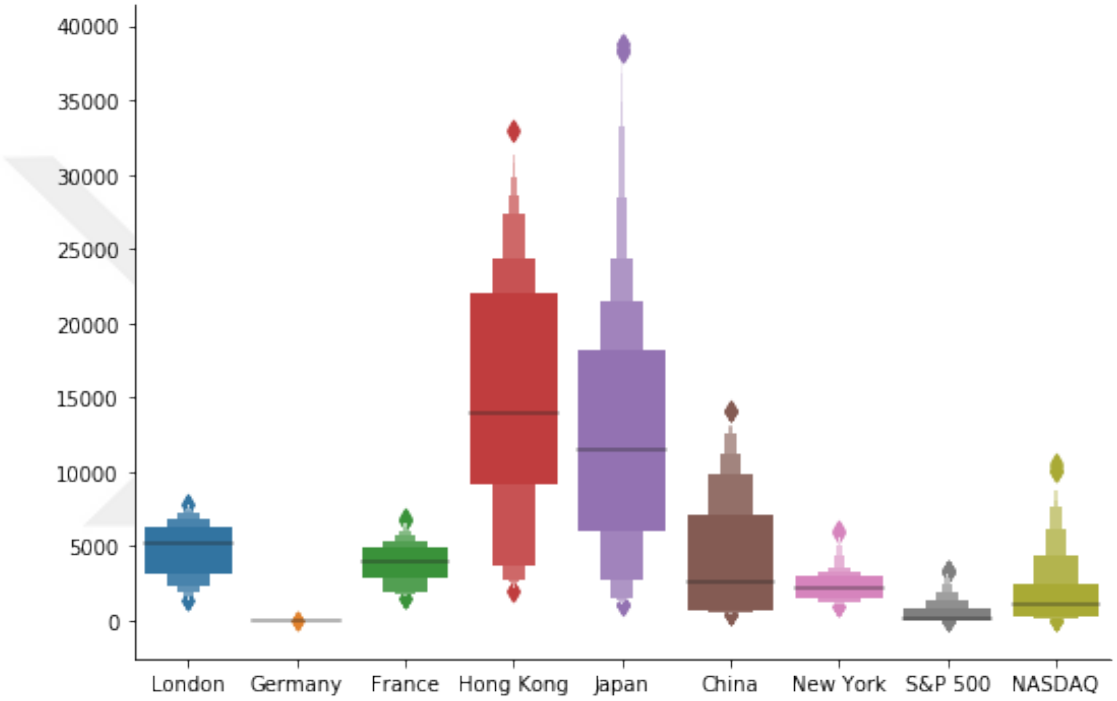


Figure 3.4: The Boxen Plots of Close Prices

The violin plots and the boxen plots of log-returns can be seen in Figure 3.5 and Figure 3.6, respectively. The first thing that stands out is that Hong Kong Stock Market Index has quite extreme data values similar to its close prices. The shapes of London, China and S&P 500 datasets are similar to each other. Also, Japan and NASDAQ have the similar shapes, on the other hand, the shapes of Germany, France, Hong Kong and New York are similar based on their densities.

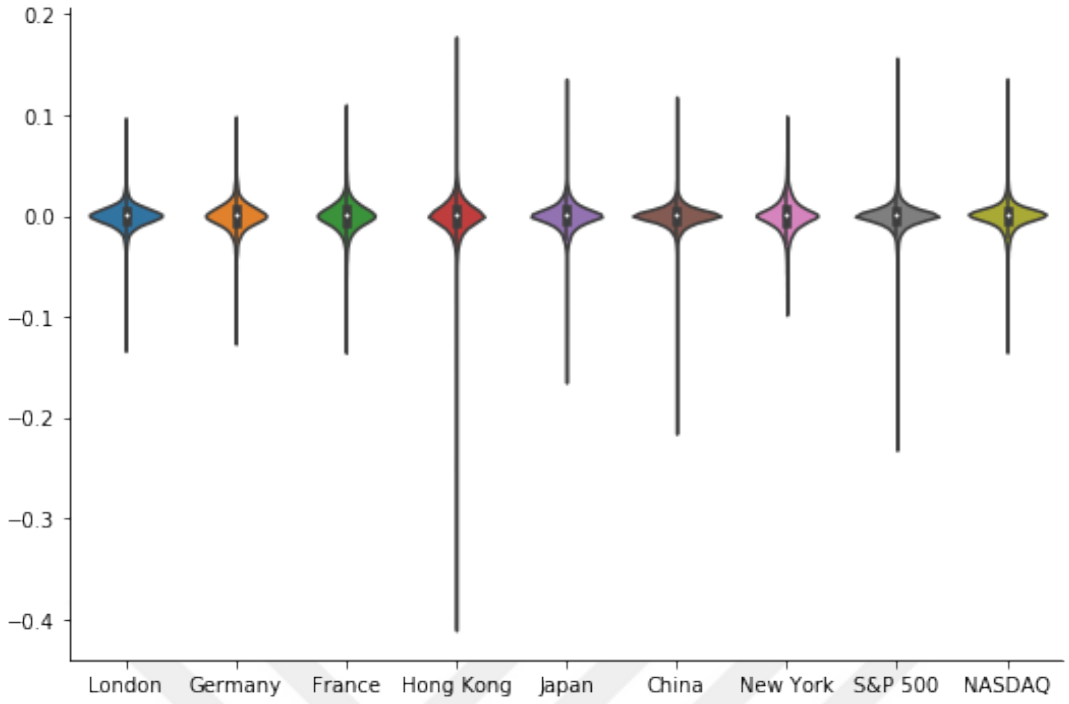


Figure 3.5: The Violin Plots of Log-Returns

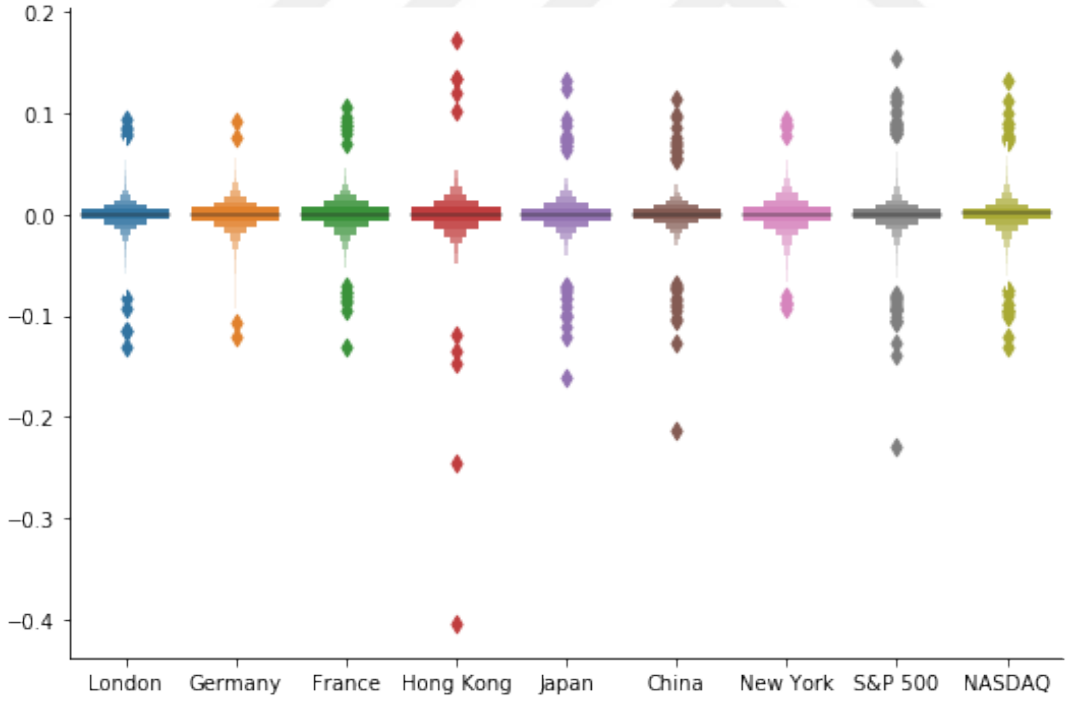


Figure 3.6: The Boxen Plots of Log-Returns

To sum up, datasets are not distributed normally according to their descriptive statistics and plots of violin & boxen.

Moreover, the cross-correlation of log-return data can be seen in Figure 3.7 as a 9x9 scatter plot. The scatter plots of the cross-correlation examines the relationship between two variables. Here, our variables are the log-returns of close prices' stock market indexes. Since cross-correlations can only examine the relationship in exactly the same length of data, and since the Germany Stock Market Index has the minimum length, so all lengths of datasets are adjusted according to length of Germany Stock Market Index. One dataset is represented in blue colour and the other one is orange. The names of the stock market indexes are located in the diagonal. Looking at the plots, it can be said that; there is an obvious relationship between S&P 500 and the NASDAQ Stock Market Indexes, and it can be explained by the fact that they are both owned by the U.S. stock market and have been affected by similar factors. Interestingly, there is a relationship between Germany and China, in other words a positive correlation. However, no such relationship has been observed in any of the other datasets.

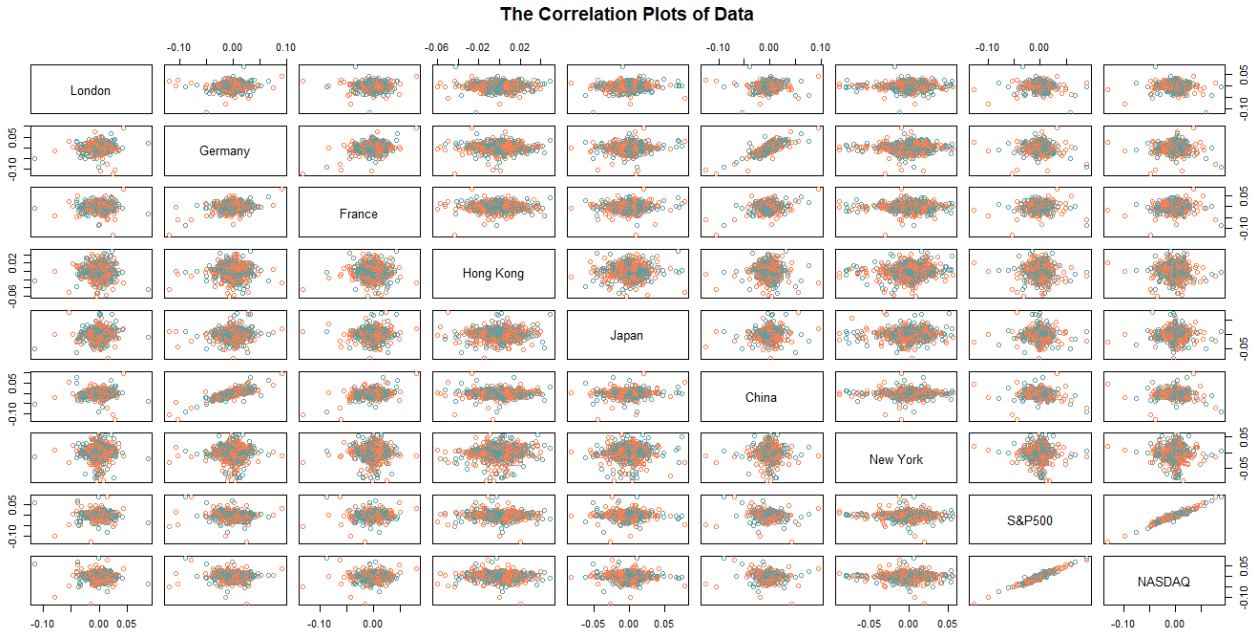


Figure 3.7: The Scatter Correlation Plots of Nine Data

3.2 Analyses of the Algorithms

3.2.1 LSTM Algorithm

LSTM Algorithm which also could be seen in Algorithm 2 is compiled in R framework with *keras* [24] and *tensorflow* [3] packages. TensorFlow which is designed for large dataset and non-homogenous environments is a ML system [3]. Accordingly, Keras as an API is developed for fast implementation can be used in TensorFlow [36].

Algorithm 2 LSTM Algorithm

1: **library** (keras, tensorflow, forecast, Metrics, ggplot2)
2: Make dataset stationary with differencing or log transforming (if necessary) [18, 21, 45]
3: Create an embedding matrix with lag order
4: Split data into training and testing
5: Normalize data with *min-max* scaling
6: Define the model
7: Compile the model with validation
8: Tune hyper-parameters
9: Fit the model on entire data
10: Invert scaling
11: Make predictions for test data and train data
12: Measure accuracy
13: Plot observed and predicted data
14: Save residuals

In the beginning, log-return data imported from Excel as a *xlsx* file is analysed. Only the "Close Prices"column is considered; therefore, log-returns are created from this column. As a result of the analysis, it is decided that data should not be cleaned to purify the outliers. Nevertheless, taking either the first or the second difference is a necessary step depending on each data. If there is back differentiation, the data is differentiated once or twice at this step. After that, a specific lag order embedding matrix is created on differentiated data to set an effective neural network. To train

the model and predict future values, data is split into two categories as training and testing. It is arranged for all nine data, as %80 for training and %20 for testing data set. Also, validation is applied first with the value of $validation\_split = 0.1$. Since LSTM requires scaling to preserved much varying range of values of data, data is normalized with *min-max* scaling between the specific range $[0, 1]$ with the following equation:

$$X_{SC} = \frac{X - X_{min}}{X_{max} - X_{min}}.$$

Then, transform is inverted for both reporting and plotting in the original scale of data. After all this, the proper model is defined with appropriate batch size and unit on *keras*. Later, the model is compiled with validation and corresponding loss function, optimizer, learning rate, epochs, batch size, and validation split on train data. Then, the model is fitted on the entire training dataset without validation and with the same arrangements. The next step is making predictions for both train and test data. Again with the invert scaling, data is in its original scale after predictions. Ultimately accuracy is measured in order to observe whether the model and the predictions are accurate or not. Likewise, the model's performance could be seen by plotting the observed and the predicted values of the model on the same graph 3.19 – 3.26. Finally, to continue with the hybrid model taking residuals is essential for using them in XGBoost Algorithm.

3.2.2 XGBoost Algorithm

To apply XGBoost, *keras* [24], *caret* (short for classification and regression training) [48] and *tidyverse* [84] packages are auxiliary mandatory packages, but the main package is *xgboost*.

Algorithm 3 XGBoost Algorithm

1: **library** (keras, tidyverse, xgboost, caret, ggplot2)
2: Create time series related features (Table 3.5 and Table 3.16)
3: Normalize necessary features
4: Place features into a data frame
5: Get correlation matrix
6: Apply Boruta Feature Selection Algorithm
7: Adjust data frame with respect to results of Boruta
8: Split data into training and testing
9: Convert train and test data into *xgb.DMatrix* format
10: Specify cross-validation method and number of folds
11: Create a hyper-parameter grid
12: Train the model with tuned hyper-parameters
13: Make predictions for train and test data
14: Measure accuracy
15: Plot observed and predicted data
16: Save residuals

Like the beginning of the LSTM algorithm, first of all, data is imported from Excel. It is considered the "Close Prices", column in which log-returns are created. It is so directly created lagged and rolling mean vectors with the specific number of lag and rolling mean, respectively, then place them into a matrix and exclude the NA values. For example, in Equation 3.1 there is a 5×4 matrix with 4 lags, and each column vector represents a lagged version of an original vector with a corresponding

lag number.

$$\begin{array}{c} \begin{matrix} lag_1 & lag_2 & lag_3 & lag_4 \end{matrix} \\ \begin{bmatrix} NA & NA & NA & NA \\ a & NA & NA & NA \\ b & a & NA & NA \\ c & b & a & NA \\ d & c & b & a \\ e & d & c & b \end{bmatrix} \\ \begin{matrix} lag_1 & lag_2 & lag_3 & lag_4 \end{matrix} \end{array} \quad \text{where } a, b, c, d, e \in \mathbb{R}. \tag{3.1}$$

Having ratio matrix, NA parts are eliminated; therefore 2×4 matrix is considered afterwards:

$$\begin{bmatrix} d & c & b & a \\ e & d & c & b \end{bmatrix} \tag{3.2}$$

where a, b, c, d, e $\in \mathbb{R}$.

Another example is rolling means. It is created with the corresponding R function, manually. It takes averages up to the given number. For instance, let the number of rolling means is $nrol = 4$, then it takes the averages of the previous three variables and the fourth variable. Then, it creates another column for this averaging process. It can be done for more than one rolling mean number, and all created columns can be gathered in one data frame or a matrix. In this thesis, the number of rolling means, and the number of lags are suitably chosen according to PACF of each data due to definition of PACF. Then, variables that may be related to the dataset are generated and put all in a data frame that will be used in XGBoost algorithm analysis. Also, Boruta Algorithm is applied to that data frame so that insignificant variables are removed. Moreover, heatmaps are considered to capture linear relationship between variables. However, adjustments of data frame are done based on Boruta Feature Selection Algorithm because it not only captures the linear relationship but also the non-linear relationship.

The regularized data frame is split into training and testing parts: %80 training and %20 testing. It is crucial to convert train and test dataset into *xgb.DMatrix* format supported by *xgboost*. Another step is specifying the cross-validation method and

number of folds to control the model.

One of the essential steps becomes regulating XGBoost hyper-parameters. For this step, a grid is created, including *the number of rounds*, *maximum depth*, *colsample by tree*, *eta*, *gamma*, *minimum child weight*, and *subsample*. Not only one single value for the hyper-parameters, but it is provided to try all the possibilities at once. Then, the model is trained with assigned hyper-parameters. According to the outputs, the hyper-parameters can be changed to obtain better results. Later, predictions are performed for both train and test data, and the accuracy is measured to utilize the model. For a better view, the observed and the predicted data are plotted to the same figure 3.19 – 3.26. At the end of this algorithm, residuals are attained to use in the LSTM hybrid model.

3.2.3 Hybrid Algorithm

As can be seen in Algorithm 4, after compiling both LSTM and XGBoost, their residuals are obtained by subtracting the predicted value from the real value. Then, XGBoost is applied to residuals of LSTM, and LSTM ,s used for residuals of XGBoost to combine the two models. After that, the single model forecasts and the residual forecasts are combined, and error measures are calculated. Finally, the evaluation criteria are reached to decide which model is more appropriate.

Algorithm 4 Hybrid Algorithm

1: Observe the forecast values from LSTM/XGBoost
2: Obtain the residuals from LSTM/XGBoost
3: Model residuals with XGBoost/LSTM
4: Combine the first model forecasts and the residual forecasts

The rest of this chapter is organized so that data and corresponding algorithms with their results are analysed in detail. Correspondingly, there are nine data and each data is handled with four algorithms.

3.3 Implementations of the Algorithms

The application of algorithms such that (1) pure LSTM, (2) XGBoost applied to residuals of LSTM, (3) LSTM applied to residuals of XGBoost and (4) pure XGBoost to nine different stock market indexes are examined in this section. Since the primary purpose is to compare the performances of the algorithms, this section is split into four by the model names. Moreover, hyper-parameters, error graphs, correlation matrices, and feature selection results for nine different datasets are presented within the context.

3.3.1 LSTM

First, the datasets downloaded from the Yahoo Finance web page and The Wall Street Journal web page as *xlsx* files are cleared of empty values in the Excel program. Then, the log-return datasets created from the stock market index's closing values are transferred to the R environment, and the analysis starts. Initially, the embedding matrix is created, and the lag order required for this matrix is selected by investigating at the significant lag in the PACF graph of data. The dataset is then normalized by using *min-max* scaling, provided that back scaling is done. Only the train set is used when scaling to avoid over-fitting. LSTM is applied separately to nine different datasets with hyper-parameters given in Table 3.4.

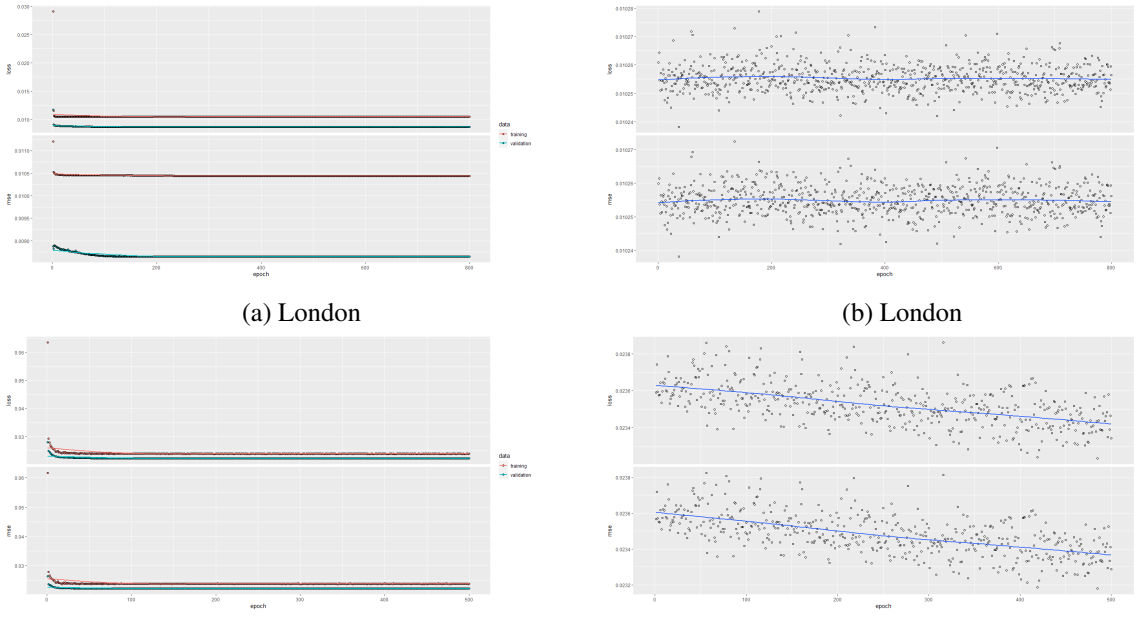
Table 3.4: Hyper-parameters used in LSTM for Nine Datasets

Hyper-parameters	**London**	**Germany**	**France**	**Hong Kong**	**Japan**	**China**	**New York**	**S&P 500**	**NASDAQ**
lag_order	7	4	5	3	6	15	16	16	18
epochs	800	500	800	800	1000	800	1000	2000	1000
batch size	32	32	32	32	32	32	128	128	64
layer_lstm(unit)	2(32,32)	1(32)	2(32,32)	2(32,32)	2(32,32)	2(32,32)	2(32,32)	2(32,32)	2(32,32)
layer_dense(unit)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)
validation_split	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
optimizer	adam	adam	adam	adam	adam	adam	adam	adam	adam
learning rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
loss	mse	mse	mse	mse	mse	mse	mse	mse	mse
metrics	mse	mse	mse	mse	mse	mse	mse	mse	mse
layer_dropout	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
L2 regularizer	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001

To choose the best hyper-parameter, the validation method is applied; therefore, the *validation_split* value is included in the model. According to the results obtained from this validation model, re-adjustments are made in the hyper-parameters, and the main LSTM model is created and applied to the whole train set. With the test set, the model's performance at values that it has never encountered hitherto is examined

in the prediction part. Also, *dropout_value* and *L2 regularization term* are added to prevent over-fitting. The model includes two LSTM layers with thirty-two internal units, one dense layer with sixteen units, and another dense layer with one unit. Only in Germany dataset there is an LSTM layer with thirty-two internal units, and the number of dense layers is the same as the other models. In line with the values in Table 3.4, the *validation-training* graph varying according to the number of epochs is given separately for each dataset in the left side of Figure 3.8, Figure 3.9 and Figure 3.10. The red colour represents the training set, and the validation is in blue. The upper side of the plot is for *Loss*, and the lower side of the plot is for *MSE*. Besides, the *Loss-MSE* scatter plots that change according to the number of epochs created for the entire dataset, not just for the validation set, is also shown in the right side of Figure 3.8 and Figure 3.9.

For validation-training plots of nine dataset there is neither under-fitting nor over-fitting situations resulting in a low or large difference between the training and validation loss, respectively. For the Loss-MSE scatter plots of nine dataset, it could be inferred that models converge.



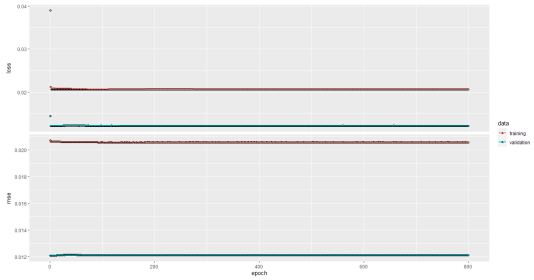
(a) London

(b) London

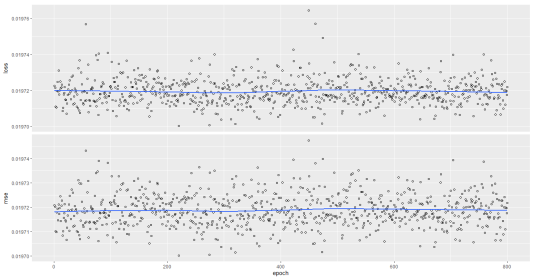
(c) Germany

(d) Germany

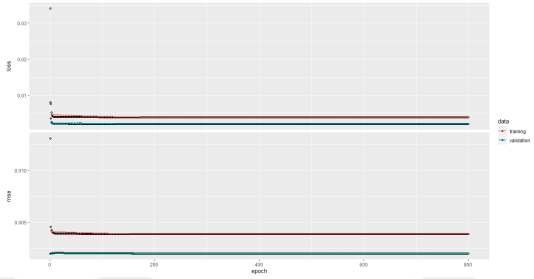
Figure 3.8: Validation-Traning (left) and Loss-MSE (right) Plots for LSTM-1



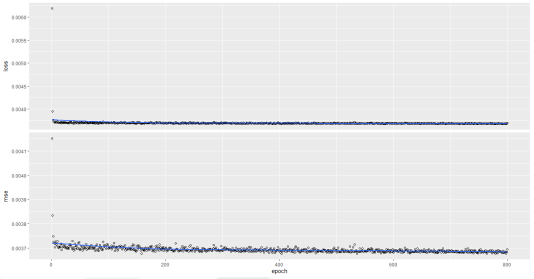
(a) France



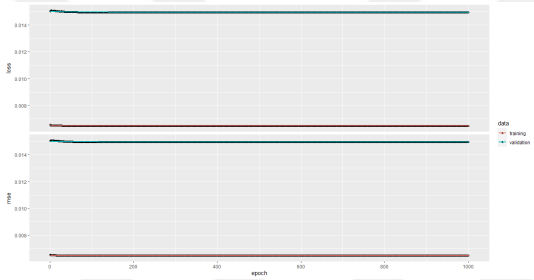
(b) France



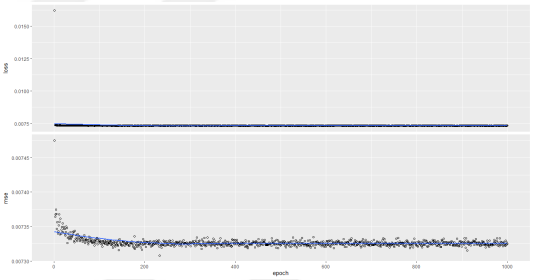
(c) Hong Kong



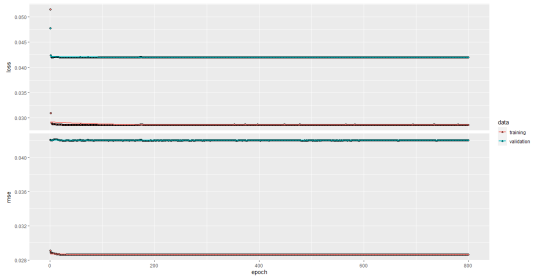
(d) Hong Kong



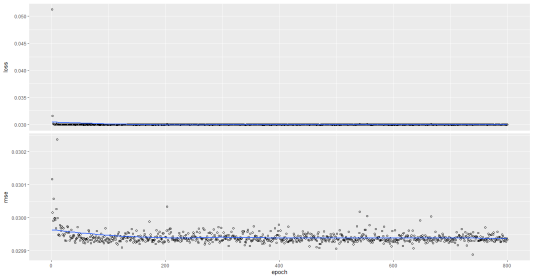
(e) Japan



(f) Japan

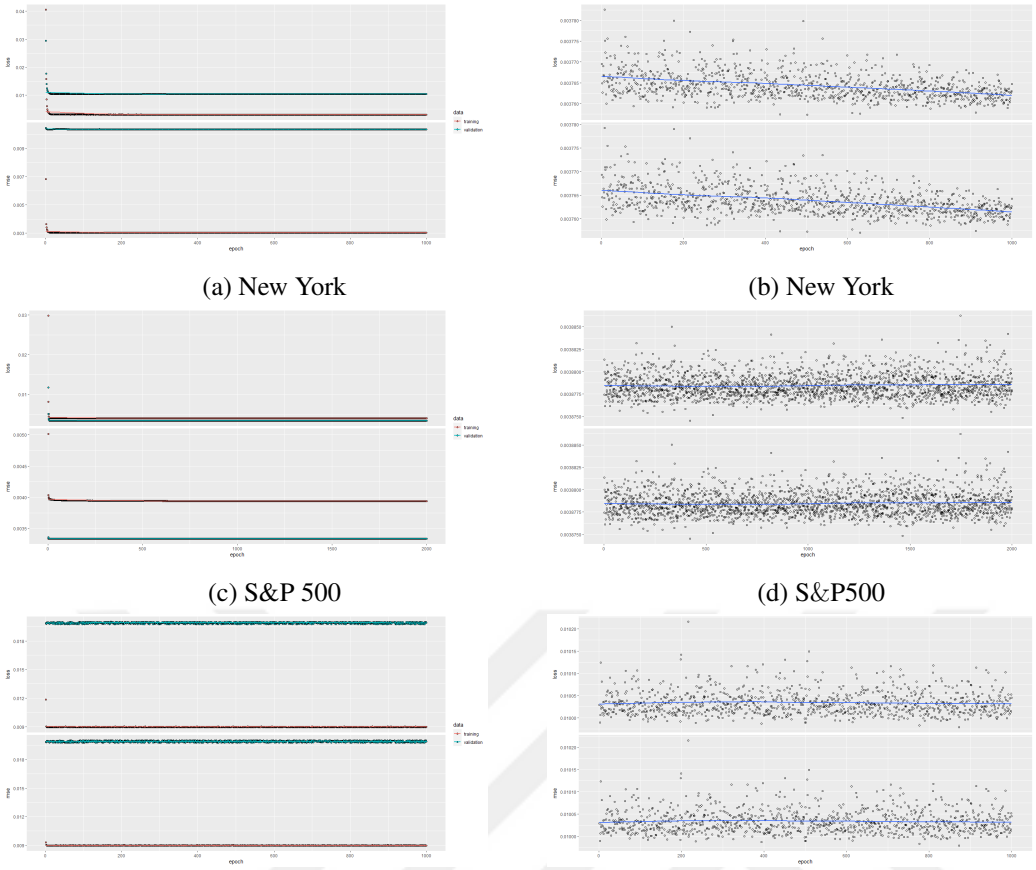


(g) China



(h) China

Figure 3.9: Validation-Traning (left) and Loss-MSE (right) Plots for LSTM-2



(a) New York (b) New York

(c) S&P 500 (d) S&P500

(e) NASDAQ (f) NASDAQ

Figure 3.10: Validation-Traning (left) and Loss-MSE (right) Plots for LSTM-3

3.3.2 XGBoost to Residuals of LSTM

It is endeavoured to increase the model accuracy by implementing the XGBoost algorithm to residuals, created by extracting predicted data from observed values from the LSTM model. This method is inspired by Zhang [90] and by Makridakis [80]. In this study, the effect of the hybridization order of different classes-LSTM (neural network algorithm) and XGBoost (tree-based algorithm)-algorithms on performance, is aimed to be examined. The data in Table 3.5 is created by considering the values that the obtained residual dataset may be related to. The lags are considered to catch AR structure and the rolling means are considered to catch MA structure in datasets. The log-returns are created from close prices but there might be a relationship between log-returns and differences of high, low, close, and open prices. So, their differences and themselves are included in the data frame. To observe change depends on time in variations, square of close prices, square of log-returns itself and for residual analysis, square of residuals of LSTM are considered. The left of the Table 3.5 entries are included because they might be relevant and to understand the non-linear structure much better. Furthermore, these entries except residuals are considered for the

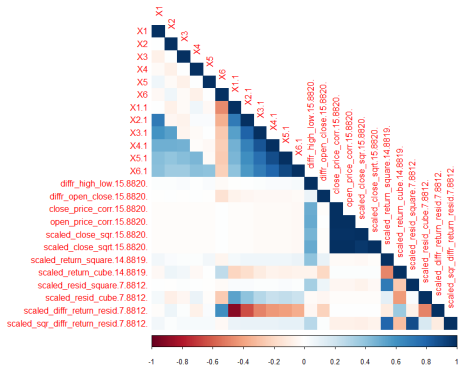
XGBoost algorithm, too. Also, the last eight data in Table 3.5 are normalized with *min-max* scaling due to the possibility of multicollinearity. After that, whole values are saved as a data frame.

Table 3.5: Correlation Matrix Entries for Residuals of LSTM

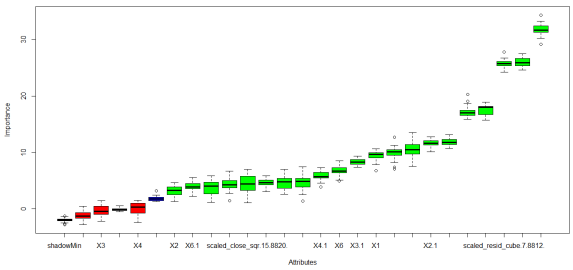
Entry	**Explanation**
X1,. . . ,Xn	Lags
X1.1,. . . ,Xn.1	Rolling means
diffr_high_low	Difference between the High and Low prices of stock market index
diffr_open_close	Difference between the Open and Close prices of stock market index
close_price_corr	Close price of data
open_price_corr	Open price of data
scaled_close_sqr	Scaled square of Close price
scaled_close_sqrt	Scaled root mean square of Close price
scaled_return_square	Scaled square of log-return
scaled_return_cube	Scaled cube of log-return
scaled_resid_sqaure	Scaled square of residuals of LSTM
scaled_resid_cube	Scaled cube of residuals of LSTM
scaled_diffr_return_resid	Scaled difference between log-return and residuals of LSTM
scaled_sqr_diffr_return_resid	Scaled square of difference between log-return and residuals of LSTM

The correlation between data in this data frame can be seen in the left sides of Figure 3.11, Figure 3.12,and Figure 3.13. In a colour scale from red to blue, the darkest red represents the negative correlation by taking the value of -1, and the darkest blue represents the positive correlation with a value of $+1$ at the x-axis. On the y-axis and at the diagonal are the names of variables with the same order as in Table 3.5. In the correlation heatmaps, obviously there is a positive correlation between rolling means. Additionally, there is a negative relationship with scaled difference between return and residuals of LSTM and other variables.

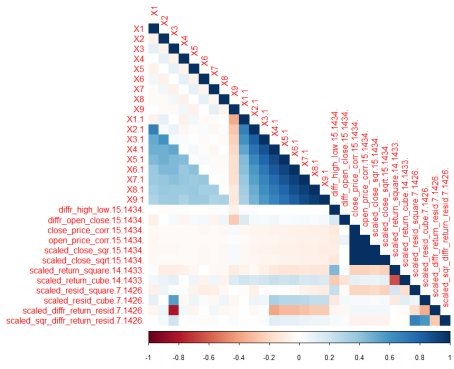
In addition, thanks to the Boruta Feature Selection Algorithm, information is obtained about which data are important and which are not. The resulting graph of the Boruta Feature Selection Algorithm can also be accessed in the right sides of Figure 3.11, Figure 3.12,and Figure 3.13. The colours in the Boruta chart; those in red represent unimportant ones, and those indicated in green represent important variables. On the x-axis, there are names of variables, and on the y-axis, importance value. Accordingly, unimportant variables are removed from the data frame before compiling the model.



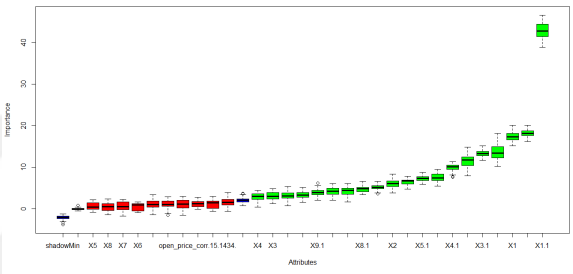
(a) Correlation Heatmap-London



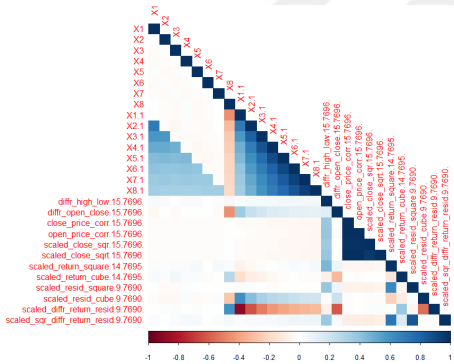
(b) Boruta-London



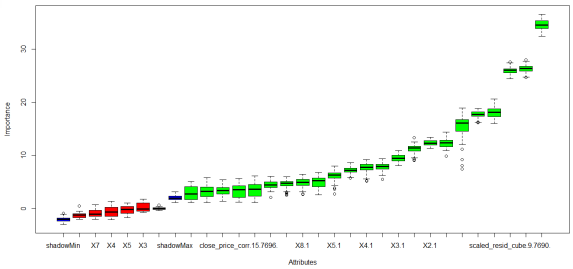
(c) Correlation Heatmap-Germany



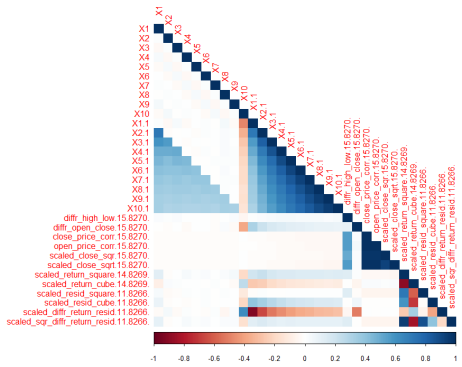
(d) Boruta-Germany



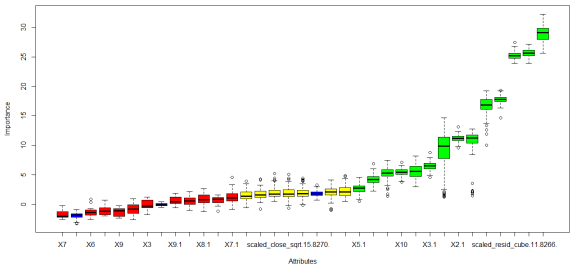
(e) Correlation Heatmap-France



(f) Boruta-France

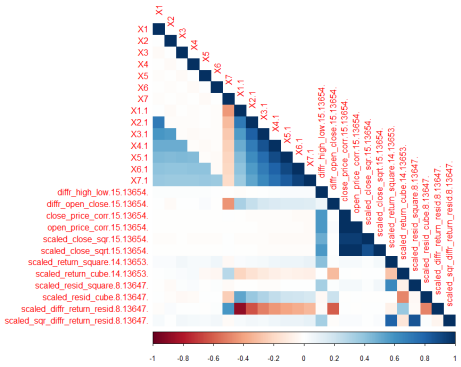


(g) Correlation Heatmap-Hong Kong

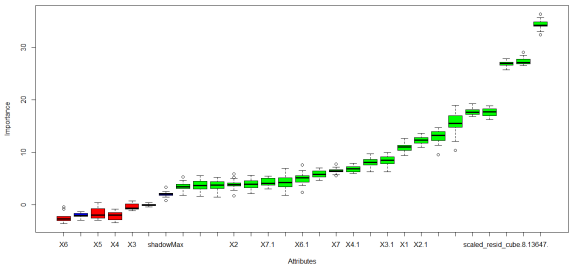


(h) Boruta-Hong Kong

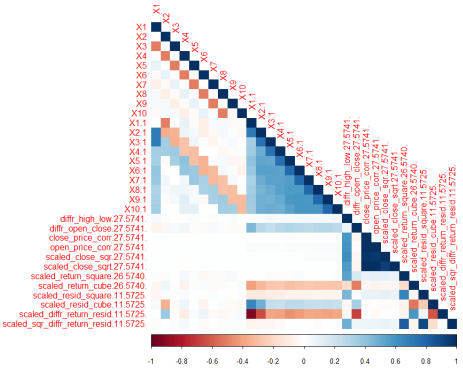
Figure 3.11: Correlation Heatmap (left) and Boruta Output (right) of Residuals of LSTM-1



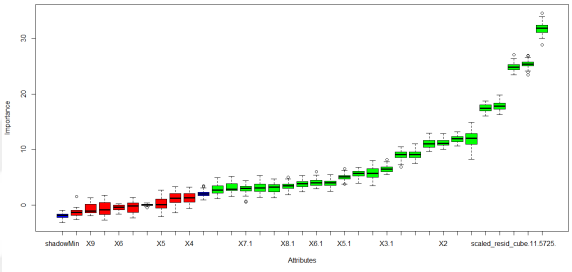
(a) Correlation Heatmap-Japan



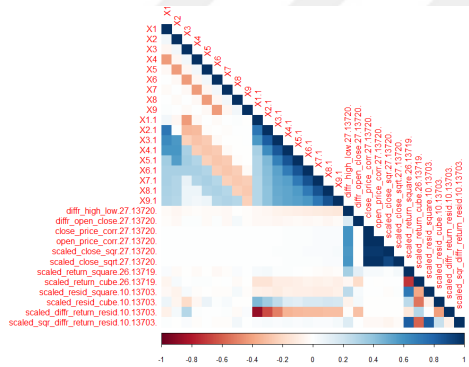
(b) Boruta-Japan



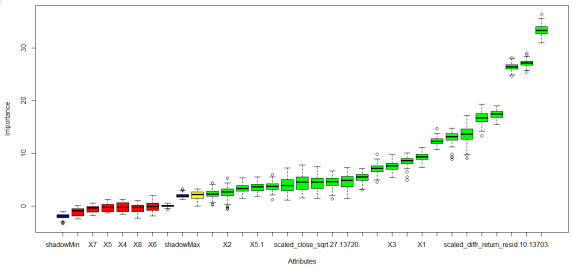
(c) Correlation Heatmap-China



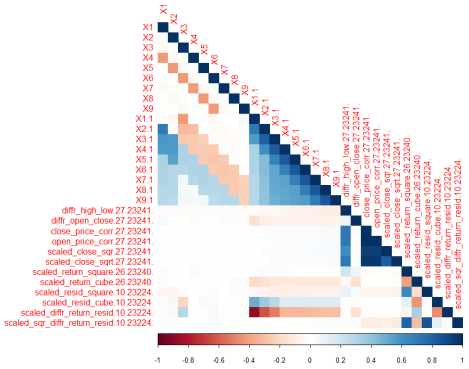
(d) Boruta-China



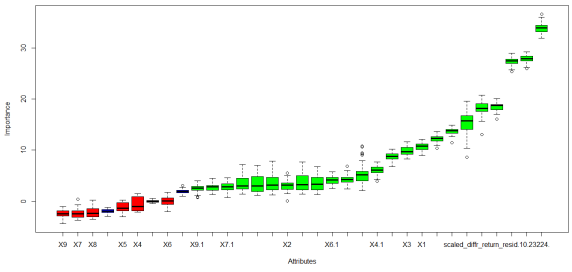
(e) Correlation Heatmap-New York



(f) Boruta-New York

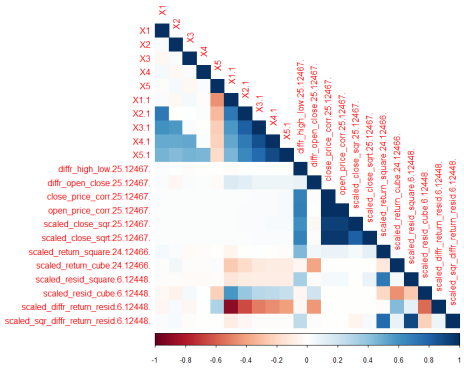


(g) Correlation Heatmap-S&P500

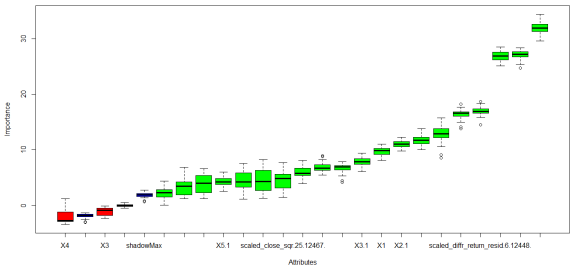


(h) Boruta-S&P 500

Figure 3.12: Correlation Heatmap (left) and Boruta Output (right) of Residuals of LSTM-2



(a) Correlation Heatmap-NASDAQ



(b) Boruta-NASDAQ

Figure 3.13: Correlation Heatmaps (left) and Boruta Outputs (right) of Residuals of LSTM-3

The outputs of Boruta for XGBoost applied to residuals of LSTM can be seen in Table 3.6 - 3.14. The outputs contains the duration of performances, the important, the unimportant and the tentative attributes. The average duration is forty minutes for nine datasets. The explanations of attributes are given in Table 3.5. Generally, lag values are often found to be insignificant for nine dataset. However, for Hong Kong Stock Market Index, not only lag values but also rolling means are evaluated as insignificant. Tentative attributes are not calculated for all datasets, but the calculated ones are included in the data frame for implementation, although they are tentative. However, the unimportant attributes are removed from the data frame.

Table 3.6: Output of Boruta-XGBoost applied to Residuals of LSTM-London

Boruta performed 29 iterations in 10.23638 mins.
21 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqr and 16 more;
3 attributes confirmed unimportant: X3, X4, X5;

Table 3.7: Output of Boruta-XGBoost applied to Residuals of LSTM-Germany

Boruta performed 64 iterations in 2.670918 mins.
20 attributes confirmed important: diffr_open_close, scaled_diffr_return_resid, scaled_resid_cube, scaled_return_cube, scaled_return_square and 15 more;
10 attributes confirmed unimportant: close_price_corr, diffr_high_low, open_price_corr, scaled_close_sqr, scaled_close_sqrt, X5, X6, X7, and X8;

Table 3.8: Output of Boruta-XGBoost applied to Residuals of LSTM-France

Boruta performed 67 iterations in 20.21529 mins.
23 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqr and 18 more;
5 attributes confirmed unimportant: X3, X4, X5, X6, X7;

Table 3.9: Output of Boruta-XGBoost applied to Residuals of LSTM-Hong Kong

Boruta performed 99 iterations in 36.15097 mins..
14 attributes confirmed important: diffr_open_close, scaled_diffr_return_resid, scaled_resid_cube, scaled_resid_square, scaled_return_cube and 9 more;
11 attributes confirmed unimportant: X10.1, X10.2, X10.3, X10.4, X10.5, X10.6, X3, X4, X5, X6 X7, X8, and X9;
7 tentative attributes left: close_price_corr, diffr_high_low, open_price_corr, scaled_close_sqr, scaled_close_sqrt and 2 more;

Table 3.10: Output of Boruta-XGBoost applied to Residuals of LSTM-Japan

Boruta performed 19 iterations in 16.81867 mins.
22 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqr and 17 more;
4 attributes confirmed unimportant: X3, X4, X5, X6;

Table 3.11: Output of Boruta-XGBoost applied to Residuals of LSTM-China

Boruta performed 64 iterations in 18.83463 mins.
24 attributes confirmed important: close_price_corr., diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqrand 19 more;
8 attributes confirmed unimportant: X3, X4, X5, X6, X7, X8, X9, and X10;

Table 3.12: Output of Boruta-XGBoost applied to Residuals of LSTM-New York

Boruta performed 99 iterations in 1.197759 hours.
23 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqrand 18 more;
6 attributes confirmed unimportant: X4, X5, X6, X7, X8 and 1 more;
1 tentative attributes left: X8.1;

Table 3.13: Output of Boruta-XGBoost applied to Residuals of LSTM-S&P 500

Boruta performed 54 iterations in 1.469398 hours.
24 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close., open_price_corr, scaled_close_sqr and 19 more;
6 attributes confirmed unimportant: X4, X5, X6, X7, X8 and X9;

Table 3.14: Output of Boruta-XGBoost applied to Residuals of LSTM-NASDAQ

Boruta performed 43 iterations in 23.07235 mins.
20 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqr and 15 more;
2 attributes confirmed unimportant: X3, X4;

Also, the number of created lags and rolling means is decided by looking at the residuals' PACF. The cross-validation method is preferred to select the best hyper-parameters for the XGBoost algorithm. The hyper-parameter table of the model XGBoost applied to the residuals obtained from LSTM of the nine datasets is shown in Table 3.15.

Table 3.15: Hyper-parameters used in XGBoost for Nine Datasets-Residuals of LSTM

Hyper-parameters	**London**	**Germany**	**France**	**Hong Kong**	**Japan**	**China**	**New York**	**S&P 500**	**NASDAQ**
number of lags	6	9	8	10	7	10	9	9	5
number of rolling means	6	9	8	10	7	10	9	9	5
number of folds	3	3	3	3	3	3	3	3	3
nrounds	2000	1000	1000	2000	2000	1500	1500	2000	2000
max_depth	6	10	10	15	15	15	10	6	10
colsample_bytree	0.9	0.8	0.9	0.9	0.9	0.9	0.8	0.9	0.9
eta	0.01	0.01	0.01	0.01	0.01	0.01	0.3	0.01	0.01
gamma	0	0	0	0	0	0	0	0	0
min_child_weight	1	1	1	1	1	1	1	1	1
subsample	1	1	1	1	1	1	1	1	1

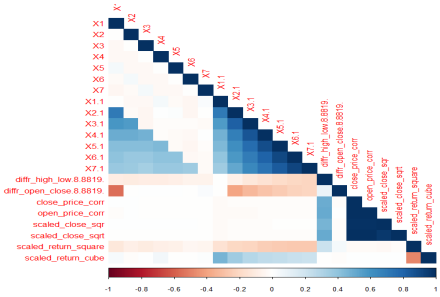
3.3.3 XGBoost

The data downloaded from the Yahoo Finance website and The Wall Street Journal website in *xlsx* format is cleared of null values first. Log-return datasets are created at the Close price of the stock market index in Excel format and transferred to the R environment. As a result of this, the data is made ready for the XGBoost application. Initially, AR values are obtained from lags and MA values from rolling means with writing appropriate functions. To decide on the number of lags and rolling means, the significant lag values of the PACF graphs of the data are examined. After the lag and rolling mean matrices, other variables assumed to be required are generated; these are displayed in Table 3.16.

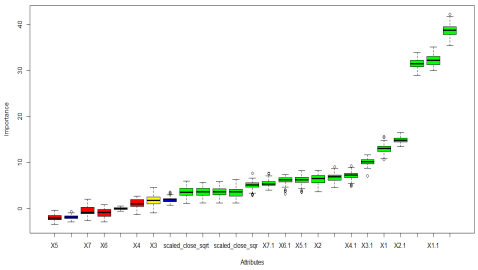
Table 3.16: Correlation Matrix Entries for XGBoost

Entry	**Explanation**
X1,. . . ,Xn	Lags
X1.1,. . . ,Xn.1	Rolling means
diffr_high_low	Difference between the High and Low prices of stock market index
diffr_open_close	Difference between the Open and Close prices of stock market index
close_price_corr	Close price of data
open_price_corr	Open price of data
scaled_close_sqr	Scaled square of Close price
scaled_close_sqrt	Scaled root mean square of Close price
scaled_return_square	Scaled square of log-return
scaled_return_cube	Scaled cube of log-return

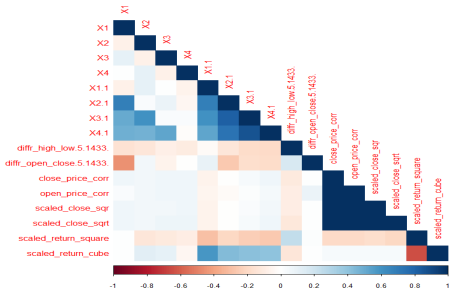
After the variables that may cause multicollinearity are normalized with *min-max* scaling, a data frame is created. The correlation matrix of this data frame is observed, and as the further step, significant and insignificant values are determined with the Boruta Feature Selection Algorithm. Graphs of Boruta Feature Selection and the Correlation Heatmap can be seen in Figure 3.14 and Figure 3.15. Correlation matrices on the left side of figures and graphics are on the right. There is a scale from dark red to dark blue in correlation matrices on the x-axis. The darkest red indicates a negative correlation with the -1 value, and the darkest blue indicates a positive correlation with the +1 value.



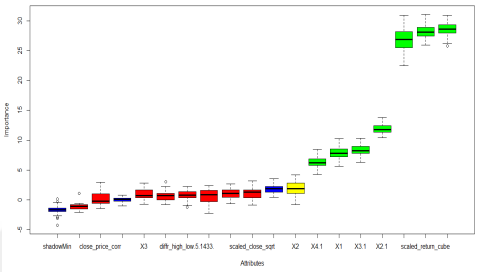
(a) Correlation Heatmap-London



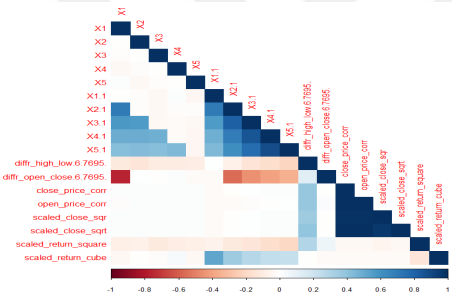
(b) Boruta-London



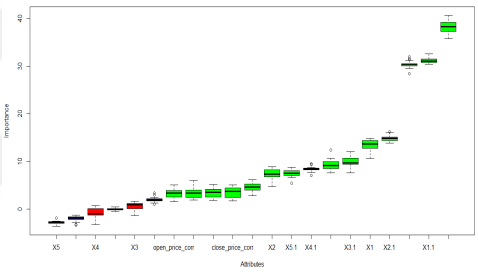
(c) Correlation Heatmap-Germany



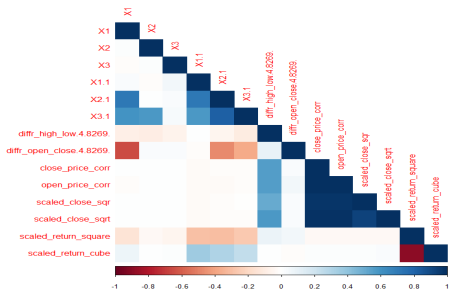
(d) Boruta-Germany



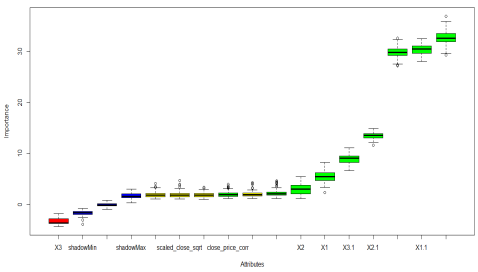
(e) Correlation Heatmap-France



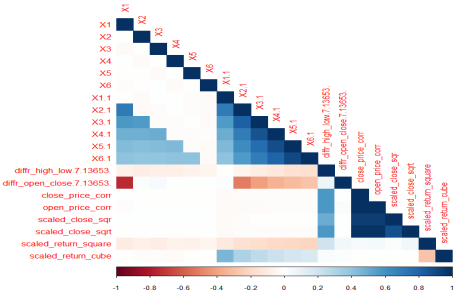
(f) Boruta-France



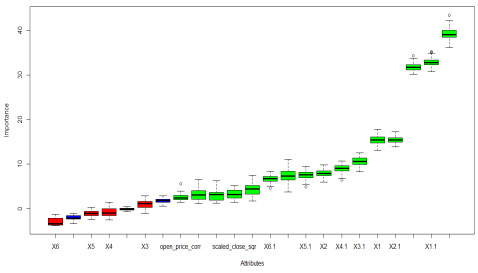
(g) Correlation Heatmap-Hong Kong



(h) Boruta-Hong Kong

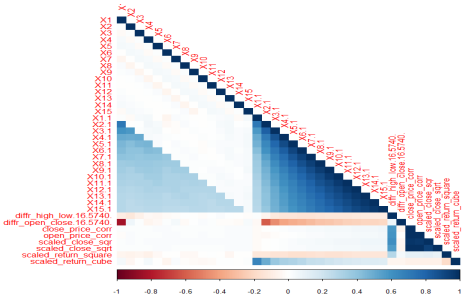


(i) Correlation Heatmap-Japan

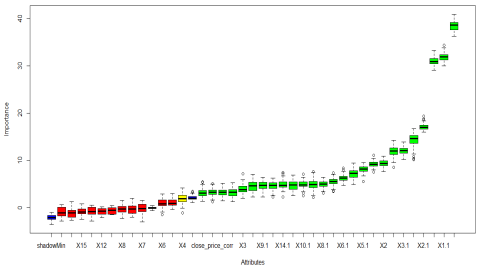


(j) Boruta-Japan

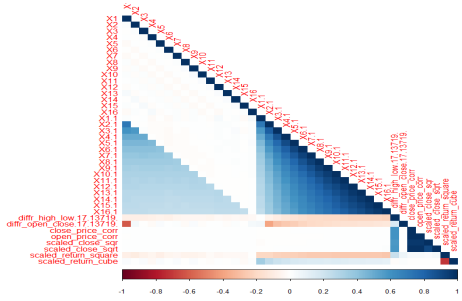
Figure 3.14: Correlation Heatmaps (left) and Boruta Outputs (right) of XGBoost-1



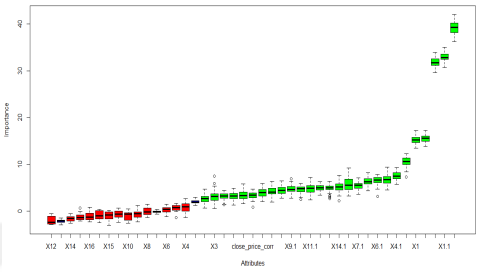
(a) Correlation Heatmap-China



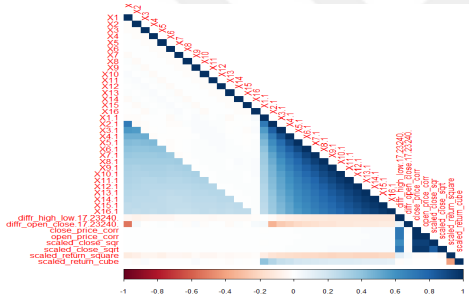
(b) Boruta- China



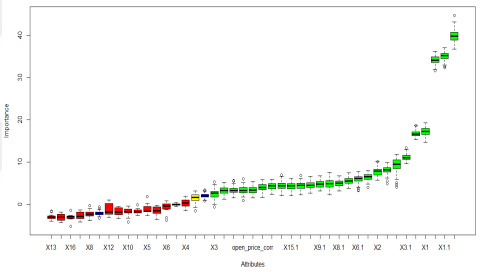
(c) Correlation Heatmap-New York



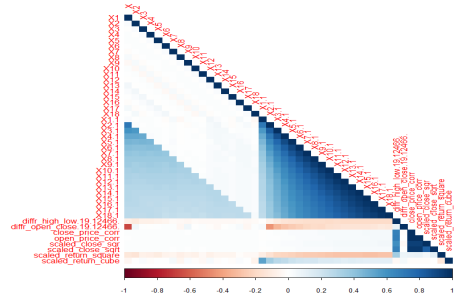
(d) Boruta-New York



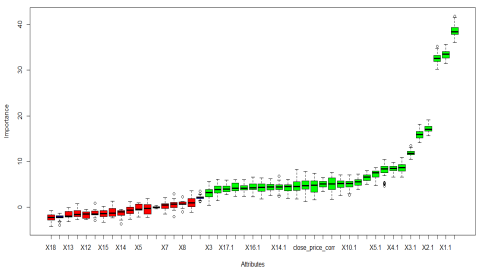
(e) Correlation Heatmap-S&P 500



(f) Boruta- S&P 500



(g) Correlation Heatmap-NASDAQ



(h) Boruta-NASDAQ

Figure 3.15: Correlation Heatmaps (left) and Boruta Outputs (right) of XGBoost-2

In Boruta charts, the green colour represents the crucial variables, and the red colour represents the insignificant variables. Variable names are on the x-axis, and their importance values are on the y-axis. A model is generated for nine separate stock market index datasets by applying the hyper-parameters in Table 3.17 to the created data frame. For the best model, the hyper-parameters shown in Table 3.17 are selected by applying the cross-validation method. Finally, the residuals required for the next

step are obtained.

Table 3.17: Hyper-parameters used in XGBoost for Nine Datasets

Hyper-parameters	**London**	**Germany**	**France**	**Hong Kong**	**Japan**	**China**	**New York**	**S&P 500**	**NASDAQ**
number of lags	7	4	5	3	6	15	16	16	18
number of rolling means	7	4	5	3	6	15	16	16	18
number of folds	3	3	3	3	3	3	3	3	3
nrounds	1000	1500	2000	1000	2000	2000	2000	1500	2000
max_depth	6	15	6	10	10	20	10	10	15
colsample_bytree	0.8	0.9	0.9	0.7	0.9	0.9	0.9	0.9	0.9
eta	0.3	0.01	0.1	0.1	0.01	0.01	0.01	0.01	0.01
gamma	0	0	0	0	0	0	0	0	0
min_child_weight	1	1	1	1	1	1	1	1	1
subsample	1	1	1	1	1	1	1	1	1

Outputs of Boruta Feature Selection Algorithm results for nine dataset are included below. It could be seen in Table 3.19 – 3.26 the important, unimportant and tentative attributes with number of iterations in minutes. Similar to Boruta results of XGBoost applied to residuals LSTM, generally, the lags except the first and second are confirmed unimportant. There are not many tentative attributes, but some of the datasets have. These tentative attributes are included in the data frame for implementation steps. Only a different result is shown in the Germany dataset which is similar to Boruta results of XGBoost applied to residuals of LSTM.

Table 3.18: Output of Boruta-XGBoost-London

Boruta performed 99 iterations in 30.69481 mins.
17 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqr and 12 more;
4 attributes confirmed unimportant: X4, X5, X6, X7;
1 tentative attributes left: X3;

Table 3.19: Output of Boruta-XGBoost-Germany

Boruta performed 99 iterations in 2.175868 mins.
7 attributes confirmed important: scaled_return_cube, scaled_return_square, X1, X1.1, X2.1 and 2 more;
8 attributes confirmed unimportant: close_price_corr, diffr_high_low, diffr_open_close open_price_corr, scaled_close_sqr,close_price_corr,scaled_close_sqrt, and X3;
1 tentative attributes left: X2;

Table 3.20: Output of Boruta-XGBoost-France

Boruta performed 39 iterations in 9.542425 mins.
15 attributes confirmed important: close_price_corr, close_sqr, close_sqrt, diffr_high_low, diffr_open_close and 10 more;
3 attributes confirmed unimportant: X3, X4, X5;

Table 3.21: Output of Boruta-XGBoost-Hong Kong

Boruta performed 99 iterations in 23.70134 mins.
9 attributes confirmed important: close_price_corr, diffr_open_close, scaled_return_cube, scaled_return_square, X1 and 4 more;
1 attributes confirmed unimportant: X3;
4 tentative attributes left: diffr_high_low, open_price_corr, scaled_close_sqr, scaled_close_sqrt;

Table 3.22: Output of Boruta-XGBoost-Japan

Boruta performed 50 iterations in 33.46352 mins.
16 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqr and 11 more;
4 attributes confirmed unimportant: X3, X4, X5, X6;

Table 3.23: Output of Boruta-XGBoost-China

Boruta performed 99 iterations in 27.89236 mins.
26 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scale_close_sqr and 21 more;
11 attributes confirmed unimportant: X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, and X15;
1 tentative attributes left: X4;

Table 3.24: Output of Boruta-XGBoost-New York

Boruta performed 57 iterations in 1.028198 hours.
27 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqr and 22 more;
13 attributes confirmed unimportant: X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15 and X16;

Table 3.25: Output of Boruta-XGBoost-S&P 500

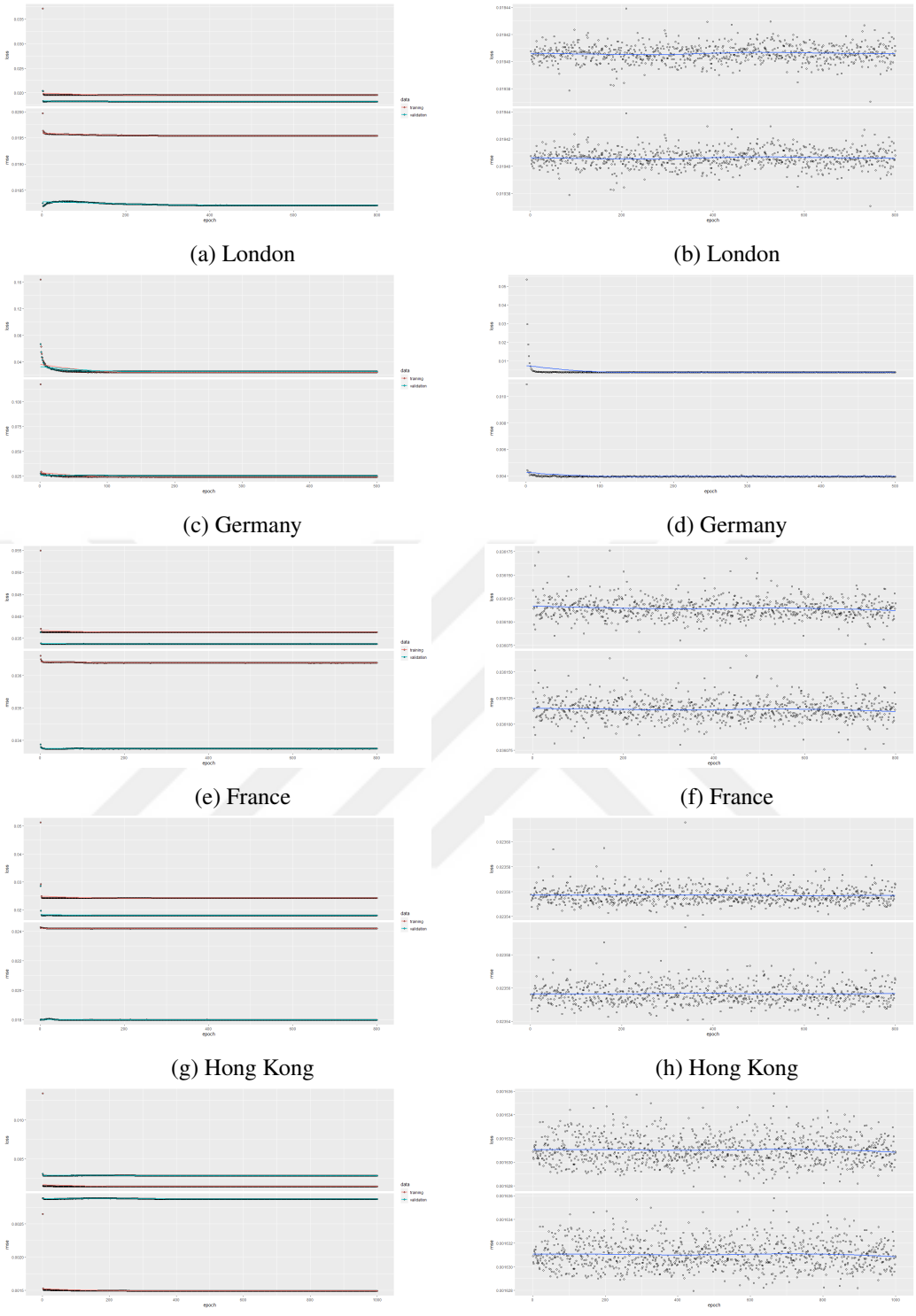
Boruta performed 99 iterations in 3.18507 hours.
26 attributes confirmed important: close_price_corr, diffr_open_close, open_price_corr, scaled_close_sqr, scaled_close_sqrt and 21 more;
13 attributes confirmed unimportant: X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15 and X16;
1 tentative attributes left: diffr_high_low;

Table 3.26: Output of Boruta-XGBoost-NASDAQ

Boruta performed 50 iterations in 47.79045 mins.
29 attributes confirmed important: close_price_corr, diffr_high_low, diffr_open_close, open_price_corr, scaled_close_sqr and 24 more;
15 attributes confirmed unimportant: X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14 X15, X16, X17 and X18;

3.3.4 LSTM to Residuals of XGBoost

The study continues by applying the LSTM model as another combination of the hybrid model to the residuals obtained from the XGBoost model. To establish the LSTM model, the residuals obtained are placed in the embedding matrix as a first step. For the lag number, the significant lag of the residuals PACF graph is taken into account due to definition of PACF which measures the linear correlation between x_t and lagged version of itself x_{t+k} with removing linear dependence of $x_{t-1}, x_{t-2}, \cdots x_{t-(k-1)}$ [40]. *Min-Max* scaling applied data becomes ready to be used in the model after this step. Dropout hyper-parameter and LSTM & Dense layers are added to avoid over-fitting. In order to prevent possible over-fitting and under-fitting conditions, the model is first tried on a validation set. According to the validation model results, it is implemented to the entire dataset with appropriate hyper-parameters. Validation-Training and Loss-MSE for the entire dataset, which vary according to the number of epochs, can be viewed in Figure 3.16 and Figure 3.17, respectively. Validation-Training plots are positioned to the left side in both figures, and Loss-MSE are to the right side. Also, in both figures, the x-axis refers to the number of epochs and the y-axis the error value. For validation-training plots of nine dataset, there is over-fitting or under-fitting. So, hyper-parameters set for validation are suitable for the model. It could be seen that they all converge in the Loss-MSE scatter plot of the model established with these hyper-parameters.



(a) London (b) London

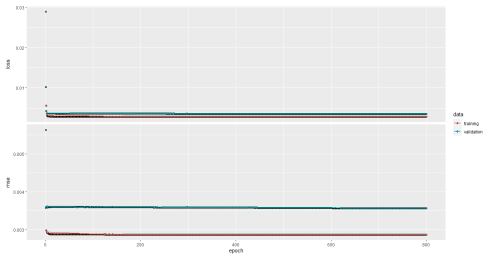
(c) Germany (d) Germany

(e) France (f) France

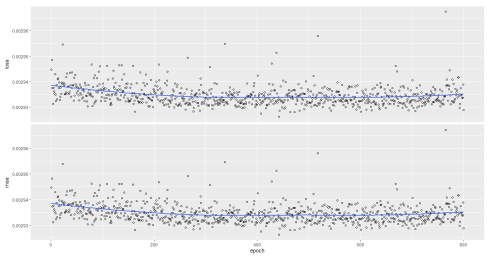
(g) Hong Kong (h) Hong Kong

(i) Japan (j) Japan

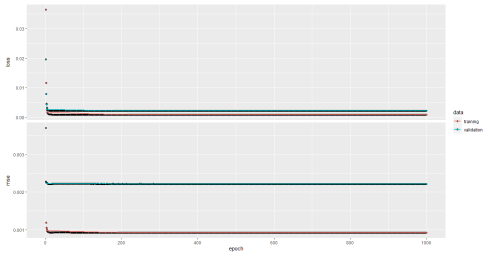
Figure 3.16: Validation-Training (left) and Loss-MSE (right) Plots for Residuals of XGBoost-1



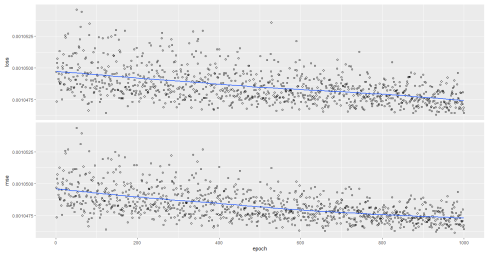
(a) China



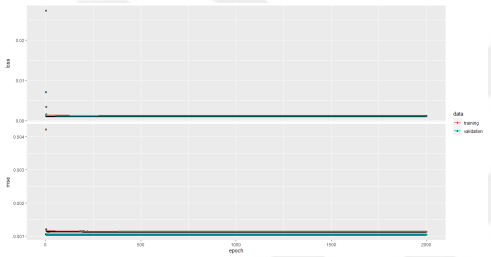
(b) China



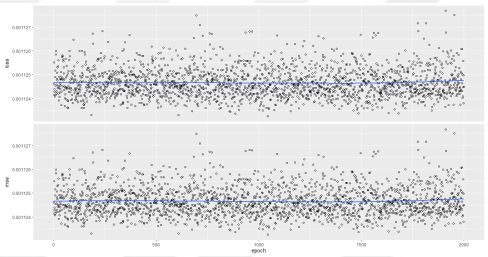
(c) New York



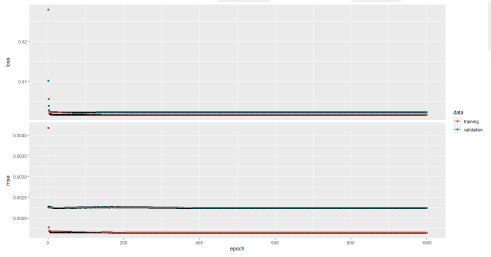
(d) New York



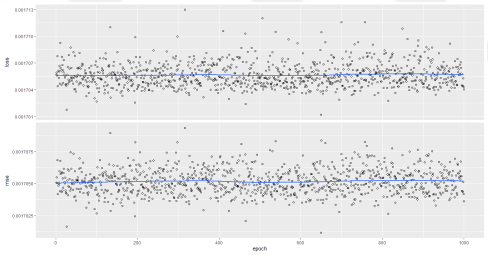
(e) S&P 500



(f) S&P 500



(g) NASDAQ



(h) NASDAQ

Figure 3.17: Validation-Training (left) and Loss-MSE (right) Plots for Residuals of XGBoost-2

The hyper-parameters used for the LSTM model applied to residuals of XGBoost can be accessed from Table 3.27.

Table 3.27: Hyper-parameters used in LSTM for Nine Datasets-Residuals of XGBoost

Hyper-parameters	London	Germany	France	Hong Kong	Japan	China	New York	S&P 500	NASDAQ
lag order	11	9	8	11	2	3	8	2	15
epochs	800	500	800	800	1000	800	1000	2000	1000
batch size	32	32	32	64	32	32	128	128	64
layer_lstm(unit)	2(32,32	1(32)	2(32,32)	2(32,32)	2(32,32)	2(32,32)	2(32,32)	2(32,32)	2(32,32)
layer_dense(unit)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)	2(16,1)
validation_split	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
optimizer	adam	adam	adam	adam	adam	adam	adam	adam	adam
learning rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
loss	mse	mse	mse	mse	mse	mse	mse	mse	mse
metrics	mse	mse	mse	mse	mse	mse	mse	mse	mse
layer_dropout	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
L2 regularizer	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001

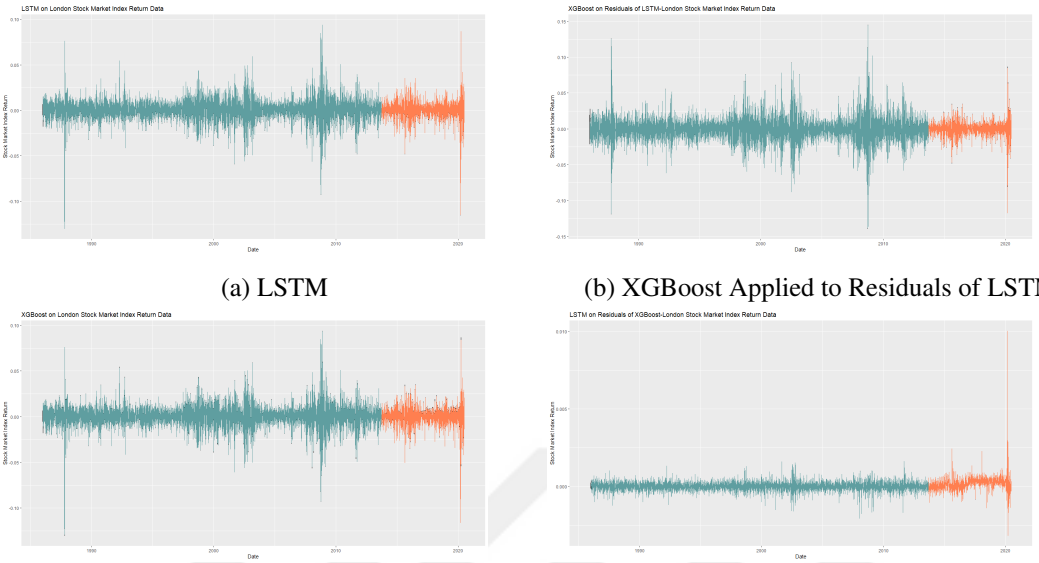
3.3.5 Results of the Algorithms

In the following sub-sections, the graphs of the observed-predicted values of the four models created for nine datasets and the error tables of these four models are given, and the results are evaluated. In this context, titles are named with datasets, and results are presented. The chart layout is the same for all datasets; in the upper left corner, is the observed-predicted graph of the LSTM result, and in the upper right corner is the XGBoost graph applied to residuals obtained from LSTM. The XGBoost application in the lower-left corner and the LSTM graph applied to the residuals obtained from XGBoost in the lower right corner. For all four plots, the date is placed on the x-axis, and values are on the y-axis. The blue colour represents the fitted train data, the testing set predictions are represented as orange, and the black lines represent the original log-return data. Besides, in error tables, model positions are the same as observed-predicted plots. Some of the error tables have *Inf* value in MAPE measure. Due to the calculation of MAPE; if there is a zero value at the bottom, then it will give zero value, which can be shown in (2.40).

3.3.5.1 London Stock Market Index Log-Return Data (FTSE 100)

The original log-return data fluctuates at the beginning and at the middle of the series. These fluctuations continues in the residuals of LSTM. However, by looking the

residuals of XGBoost it can be said that these extreme points are not visible, so it is more stable.



(a) LSTM

(b) XGBoost Applied to Residuals of LSTM

(c) XGBoost

(d) LSTM Applied to Residuals of XGBoost

Figure 3.18: Observed-Predicted Plot for London Stock Market Index Log-Return Data

— original data — predicted train — predicted test

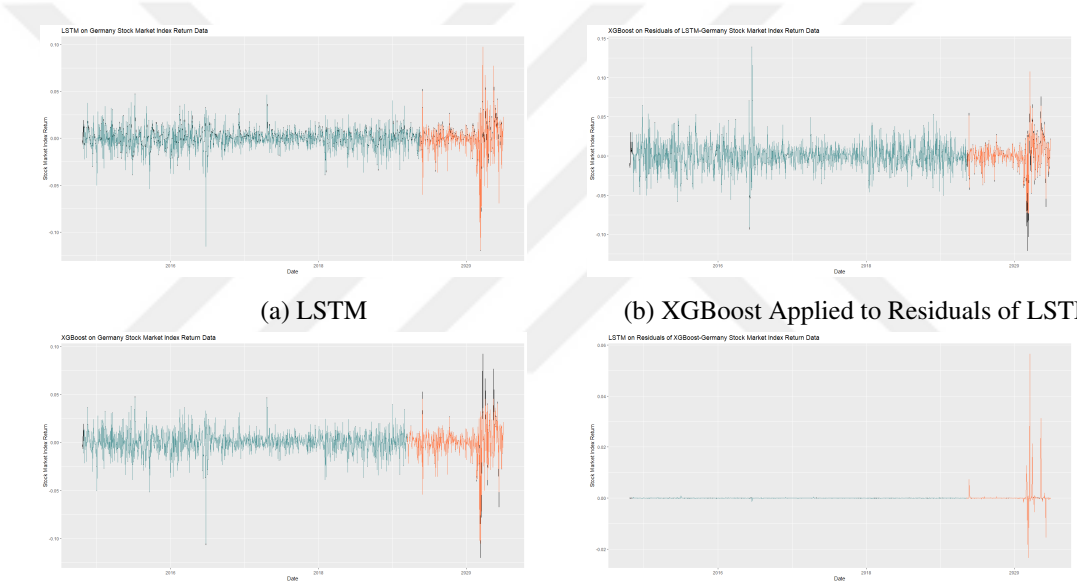
By considering these four models, the minimum RMSE, MAPE and MASE values are recorded in XGBoost applied to residuals of LSTM. In LSTM and XGBoost, MAPE values are calculated as *Inf* (infinity) because of the definition of MAPE evaluation criteria. As a result of Table 3.28, the hybrid model, especially XGBoost applied to residuals of LSTM yields the best results.

Table 3.28: Error Measures of London Stock Market Index Log-Return Data

	LSTM			XGBoost Applied to Residuals of LSTM		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
London Train	0.0163	Inf	1.0049	0.00005	0.0503	0.0025
London Test	0.0101	Inf	0.3502	0.00013	0.1634	0.0081
	XGBoost			LSTM Applied to Residuals of XGBoost		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
London Train	0.00025	Inf	0.0164	0.00036	15.2148	1.0019
London Test	0.00046	Inf	0.0317	0.00046	0.4996	0.5009

3.3.5.2 Germany Stock Market Index Log-Return Data (DAX)

In Figure 3.19b, after date 2020, there are unpredicted data with black lines. It occurs because XGBoost is not very good at predicting unseen values. In other words, the range of the unpredicted part is not in the range of the train set, so XGBoost cannot learn this pattern and therefore cannot predict. The similar issue occurs in Figure 3.19c. The upper part of the test set cannot be predicted because of the range issues. Since that part cannot be predicted, the residuals of XGBoost is large, which can also be seen in the last part of Figure 3.19d.



(a) LSTM

(b) XGBoost Applied to Residuals of LSTM

(c) XGBoost

(d) LSTM Applied to Residuals of XGBoost

Figure 3.19: Observed-Predicted Plot for Germany Stock Market Index Log-Return Data

— original data — predicted train — predicted test

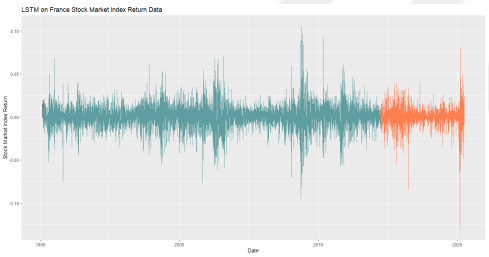
The minimum RMSE and MASE values for both train and test datasets are evaluated with XGBoost. Nevertheless, the minimum MAPE values are evaluated in XGBoost applied to residuals of LSTM. Table 3.29 shows that the hybrid model does not yield results better; pure XGBoost model performs the best results for Germany stock market index log-return data.

Table 3.29: Error Measures of Germany Stock Market Index Log-Return Data

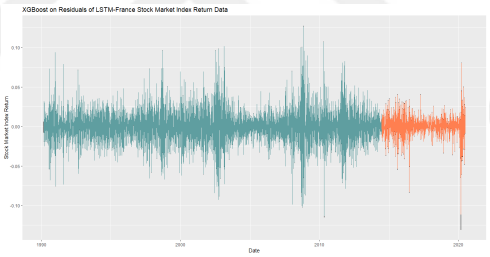
	LSTM			XGBoost Applied to Residuals of LSTM		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
Germany Train	0.0171	Inf	0.9963	0.0002	0.0455	0.0074
Germany Test	0.0210	Inf	0.3259	0.0056	0.1927	0.0791
	XGBoost			LSTM Applied to Residuals of XGBoost		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
Germany Train	0.000072	Inf	0.0030	0.00010	14.0902	0.9834
Germany Test	0.0042	Inf	0.0371	0.0045	0.6153	0.2583

3.3.5.3 France Stock Market Index Log-Return Data (FCHI)

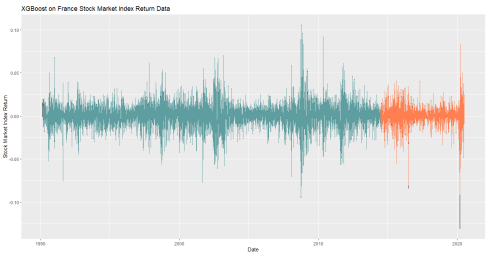
For three plots; Figure 3.20a, 3.20b and 3.20c, there are some extreme values at the similar times but well predicted. Figure 3.20d, at the end of the test set, there is a specific outlier. It comes from unsuccessful XGBoost prediction at the end of the test set due to range issues of XGBoost mentioned in the previous subsection with Germany data.



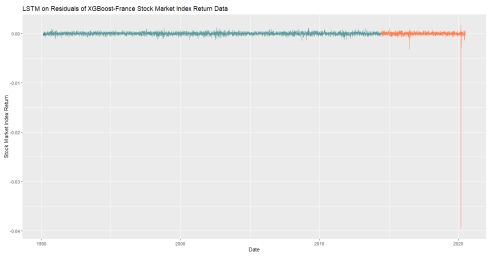
(a) LSTM



(b) XGBoost Applied to Residuals of LSTM



(c) XGBoost



(d) LSTM Applied to Residuals of XGBoost

Figure 3.20: Observed-Predicted Plot for France Stock Market Index Log-Return Data

— original data — predicted train — predicted test

Considering four models, the minimum RMSE, MAPE, and MASE values occur at XGBoost applied to residuals of LSTM. It implies that the hybridization of LSTM & XGBoost performs better than pure models. Also, it can be mentioned about the hy-

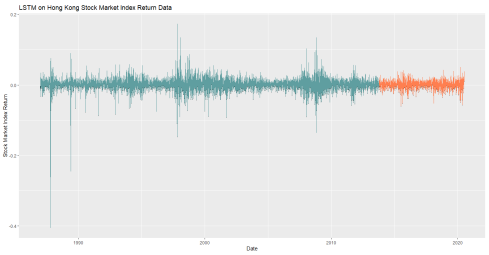
bridization order so that XGBoost applied to residuals of LSTM is better than LSTM applied to residuals of XGBoost. In other words, the best result is obtained by first applying LSTM to log-return data and then implementing XGBoost to the residuals of LSTM.

Table 3.30: Error Measures of France Stock Market Index Log-Return Data

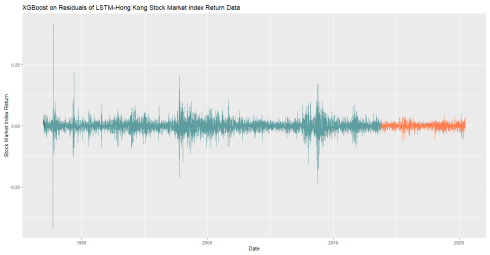
	LSTM			XGBoost Applied to Residuals of LSTM		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
France Train	0.0197	Inf	0.9769	0.00007	0.0722	0.0023
France Test	0.0127	0.5087	0.3488	0.00053	0.0319	0.0104
	XGBoost			LSTM Applied to Residuals of XGBoost		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
France Train	0.0002	Inf	0.0126	0.00034	12.6647	0.9973
France Test	0.0010	0.0960	0.0179	0.0010	0.5036	0.3336

3.3.5.4 Hong Kong Stock Market Index Log-Return Data (HSI)

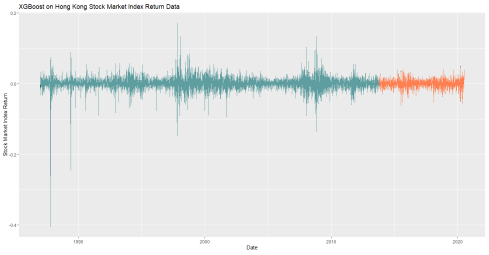
There are specific two extreme values at the original Hong Kong log-return data, and it can be seen at the beginning of Figures 3.21a and 3.21c. The result of poor prediction in Figure 3.21b also shows two extreme values at the beginning of the data. Interestingly, in Figure 3.21d, although the general structure of data looks stationary and well predicted, there is an outlier at the end of the test data.



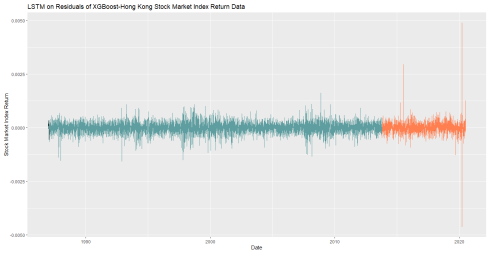
(a) LSTM



(b) XGBoost Applied to Residuals of LSTMa



(c) XGBoost



(d) LSTM Applied to Residuals of XGBoost

Figure 3.21: Observed-Predicted Plot for Hong Kong Stock Market Index Log-Return Data

— original data — predicted train — predicted test

As a result of error values, the minimum RMSE, MAPE, and MASE values are obtained from the XGBoost applied to the residuals of LSTM. Even though the opposite order of hybridization does not make any reasonable contribution to decrease MASE errors, XGBoost applied to the residuals of LSTM decreases errors significantly. So, it concludes that hybridization gives better results.

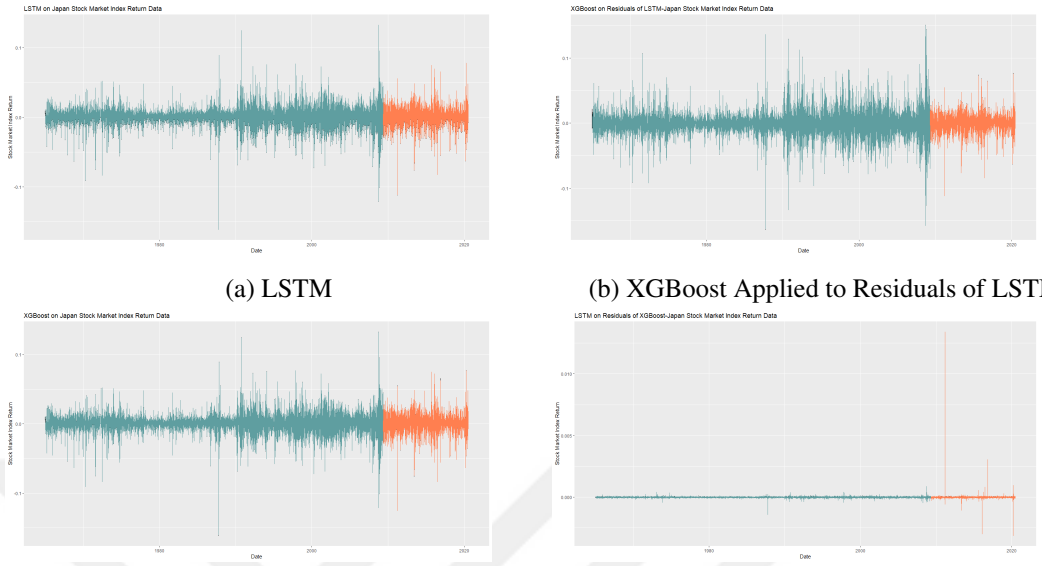
Table 3.31: Error Measures of Hong Kong Stock Market Index Log-Return Data

	LSTM			XGBoost Applied to Residuals of LSTM		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
Hong Kong Train	0.0248	Inf	1.0225	0.00007	0.0217	0.0021
Hong Kong Test	0.0114	1.2271	0.3442	0.00012	0.0587	0.0070
	XGBoost			LSTM Applied to Residuals of XGBoost		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
Hong Kong Train	0.00024	Inf	0.0112	0.00035	6.9172	1.0061
Hong Kong Test	0.00030	0.5079	0.0156	0.00030	0.5001	0.3481

3.3.5.5 Japan Stock Market Index Log-Return Data (N225)

In Figure 3.22a, Figure 3.22b and Figure 3.22c between year 1989-2010, Japan log-return data fluctuates much more than the rest of data. In Figure 3.22d, there are a few extreme values on the test dataset. This is due to the fact that the difference

between observed data and predicted with XGBoost data is high at that point. So, the big residuals occur.



(a) LSTM

(b) XGBoost Applied to Residuals of LSTM

(c) XGBoost

(d) LSTM Applied to Residuals of XGBoost

Figure 3.22: Observed-Predicted Plot for Japan Stock Market Index Log-Return Data

— original data — predicted train — predicted test

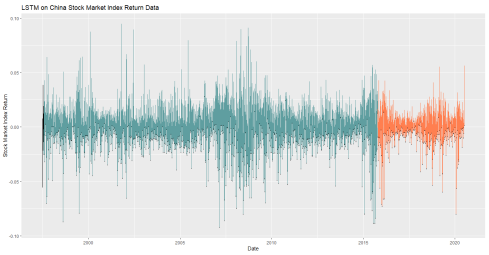
The minimum MAPE values examined from XGBoost are applied to residuals of LSTM, but for the other error measures, the results are different. While the minimum RMSE value for train occurs at XGBoost, for test data, it is in XGBoost applied to the residuals of LSTM. Furthermore, the minimum MASE value for the train is calculated with XGBoost applied to residuals of LSTM, but the minimum MASE value for the test set is at XGBoost.

Table 3.32: Error Measures Japan Stock Market Index Log-Return Data

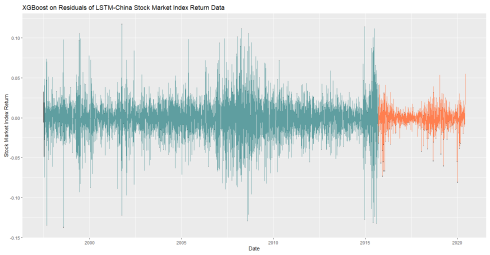
	LSTM			XGBoost Applied to Residuals of LSTM		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
Japan Train	0.0176	Inf	1.0053	0.00005	0.0272	0.0020
Japan Test	0.0135	Inf	0.3413	0.00008	0.0313	0.0036
	XGBoost			LSTM Applied to Residuals of XGBoost		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
Japan Train	0.00005	Inf	0.0025	0.00006	6.9203	0.9696
Japan Test	0.00028	Inf	0.0032	0.00028	0.9117	0.3156

3.3.5.6 China Stock Market Index Log-Return Data (SSE)

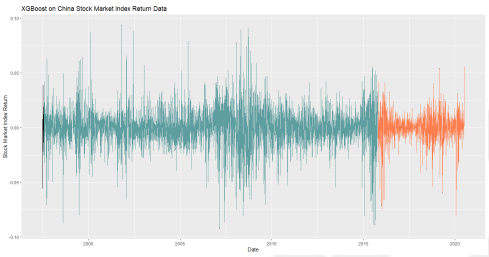
Figure 3.23a, Figure 3.23b, and Figure 3.23c have the similar structure and fluctuates more especially between the year 2006-2010. Figure 3.23d, the residuals of XGBoost fluctuates less with respect to others, but there are few extreme values.



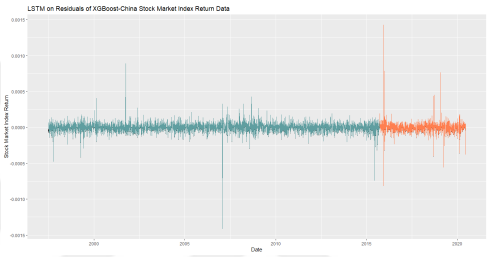
(a) LSTM



(b) XGBoost Applied to Residuals of LSTM



(c) XGBoost



(d) LSTM Applied to Residuals of XGBoost

Figure 3.23: Observed-Predicted Plot for China Stock Market Index Log-Return Data

— original data — predicted train — predicted test

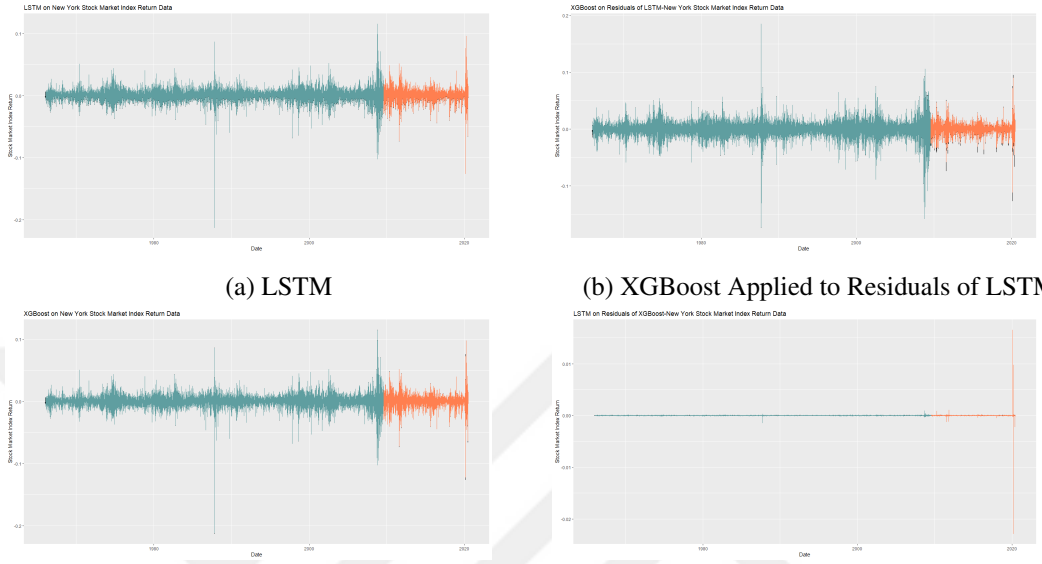
The minimum RMSE value is calculated with XGBoost for both the train and test datasets. Also, the minimum MAPE and MASE values for the test dataset are calculated with XGBoost. However, for the training dataset, MAPE and MASE values get their minimum values with XGBoost applied to the residuals of LSTM. However, there is no a big gap between the error values between XGBoost and XGBoost applied to residuals of LSTM.

Table 3.33: Error Measures of China Stock Market Index Log-Return Data

	LSTM			XGBoost Applied to Residuals of LSTM		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
China Train	0.	Inf	1.0151	0.00007	0.0257	0.0020
China Test	0.	0.5629	0.3332	0.00018	0.0577	0.0078
	XGBoost			LSTM Applied to Residuals of XGBoost		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
China Train	0.000062	Inf	0.0025	0.000088	11.3595	0.9956
China Test	0.000082	0.0167	0.0038	0.000082	0.5877	0.3361

3.3.5.7 New York Stock Market Index Log-Return Data (NYA)

New York data does not have many more fluctuations and extreme values. Especially, Figure 3.24d is pretty stable around zero. There is just an outlier at the end of the test data. This is due of the difference between XGBoost observed and predicted data.



(a) LSTM

(b) XGBoost Applied to Residuals of LSTMa

(c) XGBoost

(d) LSTM Applied to Residuals of XGBoost

Figure 3.24: Observed-Predicted Plot for New York Stock Market Index Log-Return Data

— original data — predicted train — predicted test

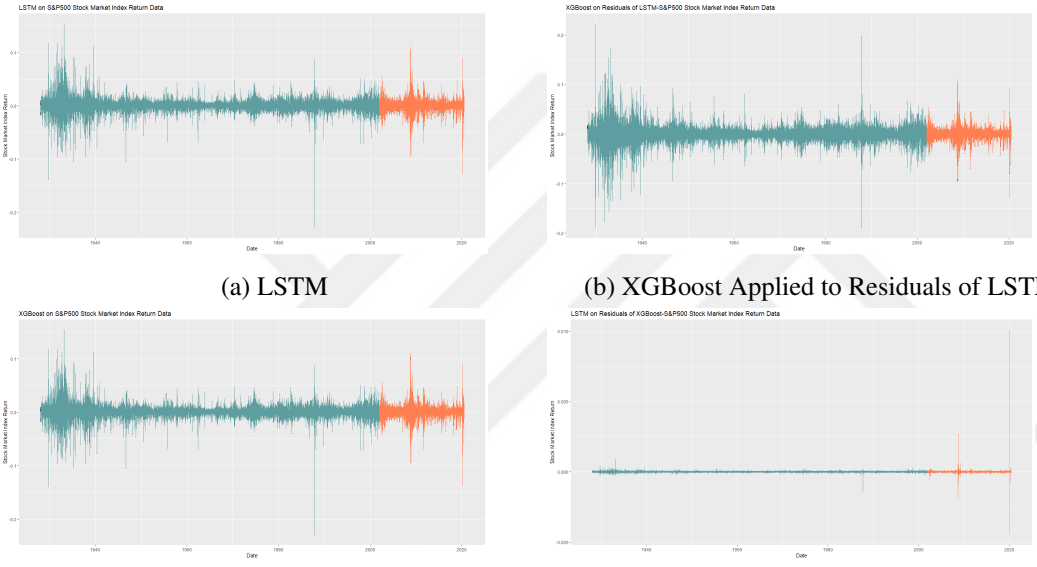
The minimum RMSE and MASE values occur at the results of XGBoost for both train and test datasets. However, the minimum MAPE value for both train and test datasets occurs at the results of XGBoost applied to residuals of LSTM. Here, it is very obvious that the pure XGBoost model performs better than the other methods. Thus, hybridization does not make results better for this dataset.

Table 3.34: Error Measures of New York Stock Market Index Log-Return Data

	LSTM			XGBoost Applied to Residuals of LSTM		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
New York Train	0.0141	Inf	1.0579	0.00017	0.1672	0.0097
New York Test	0.0112	Inf	0.3376	0.00101	0.1470	0.0244
	XGBoost			LSTM Applied to Residuals of XGBoost		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
New York Train	0.00004	Inf	0.0026	0.00005	7.1845	1.0057
New York Test	0.00061	Inf	0.0049	0.00061	0.7612	0.2977

3.3.5.8 S&P 500 Stock Market Index Log-Return Data (GSPC)

In the early 1940s, S&P 500 log-return data has some fluctuations, but after the 1940s, it stays stationary. Similar to New York data, there are few extreme values both in the train and test dataset.



(a) LSTM

(b) XGBoost Applied to Residuals of LSTM

(c) XGBoost

(d) LSTM Applied to Residuals of XGBoost

Figure 3.25: Observed-Predicted Plot for S&P 500 Stock Market Index Log-Return Data

— original data — predicted train — predicted test

The minimum MAPE values for both train and test datasets are examined from XGBoost applied to residuals of LSTM. The minimum MASE value for the train set and the minimum RMSE value for the test set are obtained with XGBoost applied to residuals of LSTM. However, the minimum RMSE value for the train and the minimum MASE value for the test are calculated with XGBoost. Error values are very close to each other, so drawing a conclusion is quite hard.

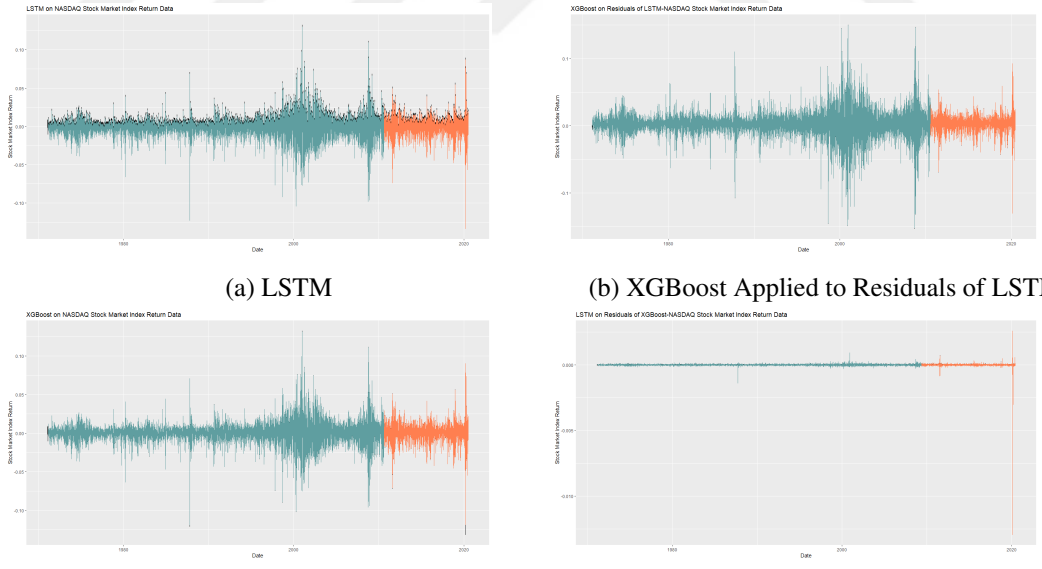
Table 3.35: Error Measures S&P 500 Stock Market Index Log-Return Data

	LSTM			XGBoost Applied to Residuals of LSTM		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
S&P 500 Train	0.0168	Inf	1.0466	0.00005	0.0179	0.0021
S&P 500 Test	0.0124	Inf	0.3303	0.00016	0.0246	0.0058
	XGBoost			LSTM Applied to Residuals of XGBoost		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
S&P 500 Train	0.00004	Inf	0.0024	0.0002	10.4609	1.0062
S&P 500 Test	0.00017	Inf	0.0028	0.0003	0.6913	0.3377

Table 3.36: S&P 500 Stock Market Index Log-Return Data Error Measures

3.3.5.9 NASDAQ Stock Market Index Log-Return Data (IXIC)

Even though NASDAQ log-return data is pretty stable, there are some extreme values, and especially around the year, 2000 fluctuations are too many. This situation does not affect predictions negatively. There is an unsuccessful prediction at XGBoost at the end of the test data with black lines. Due to the fact that it is out of the range of the XGBoost train dataset. Therefore, the residual of that point turns to be quite large, as shown in Figure 3.26d.



(a) LSTM

(b) XGBoost Applied to Residuals of LSTM

(c) XGBoost

(d) LSTM Applied to Residuals of XGBoost

Figure 3.26: Observed-Predicted Plot for NASDAQ Stock Market Index Log-Return Data

— original data — predicted train — predicted test

The minimum MAPE and MASE values are obtained from XGBoost applied to resid-

uals of LSTM. Also, within this method, the minimum RMSE value for test data is calculated; however, the minimum RMSE value for train data is obtained from XGBoost. Still, the values are pretty close to each other. So, we may conclude that hybridization makes results better.

Table 3.37: Error Measures of NASDAQ Stock Market Index Log-Return Data

	LSTM			XGBoost Applied to Residuals of LSTM		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
NASDAQ Train	0.0180	Inf	1.0788	0.000049	0.0226	0.0022
NASDAQ Test	0.0124	Inf	0.3615	0.000068	0.0210	0.0030
	XGBoost			LSTM Applied to Residuals of XGBoost		
	RMSE	MAPE	MASE	RMSE	MAPE	MASE
NASDAQ Train	0.00004	Inf	0.0029	0.00006	7.8414	0.9832
NASDAQ Test	0.00028	Inf	0.0038	0.00026	0.6066	0.3159

Summary of Results

To conclude, MAPE for nine of the train datasets, MASE for seven, RMSE for three, the smallest values are obtained by applying XGBoost to residuals of LSTM. MAPE for eight of the nine test datasets, MASE for four, RMSE for six, the smallest values are recorded in applying XGBoost to residuals of LSTM. Moreover, MAPE for one of the nine datasets, MASE for five, RMSE for three, the smallest values are obtained in applying XGBoost. MAPE for none of the nine test datasets, MASE for two, RMSE for six obtained the smallest values in applying XGBoost. For London, France, Hong Kong and NASDAQ Stock Market Indexes, the best results are recorded in XGBoost applied to residuals of LSTM. However, for Germany and New York dataset, XGBoost itself gives the best results. For the rest of the datasets, Japan, China and S&P500, XGBoost applied to residuals of LSTM gives the best results in train sets, but in test sets XGBoost gives the best result. Overall, according to MASE results, it can be concluded that the hybridization of LSTM and XGBoost, specifically, applying XGBoost to residuals of LSTM, gives better results. The summary of the results could be seen in Table 3.38. The highlighted values are the minimum MASE values of each dataset.

If datasets are considered as Europe, Asia and U.S., it can be seen clearly that there is not an exact aggregation. In two-thirds of Europe datasets, XGBoost applied to resid-

uals of LSTM showed better results. In all train datasets of Asia markets, XGBoost applied to residuals of LSTM, in two-thirds test datasets of Asia markets XGBoost gives the minimum MASE values. Among the American markets, one dataset gave the best results with XGBoost, while another gave the best results with XGBoost applied to residuals of LSTM. The last one reached the minimum MASE values with XGBoost applied to residuals of LSTM in the training set and XGBoost itself in the test set. Therefore, it is hard to make geographical inferences between stock market indexes and the algorithms.

Table 3.38: Comparison of Four Models on Nine Dataset According to MASE

		LSTM	**XGBoost Applied to Residuals of LSTM**	**XGBoost**	**LSTM Applied to Residuals of XGBoost**
London	**Train**	1.0049	0.0025	0.0164	1.0019
	Test	0.3502	0.0081	0.0317	0.5009
Germany	**Train**	0.9963	0.0074	0.0030	0.9834
	Test	0.3259	0.0791	0.0371	0.2583
France	**Train**	0.9769	0.0023	0.0126	0.9973
	Test	0.3488	0.0104	0.0179	0.3336
Hong Kong	**Train**	1.0225	0.0020	0.0112	1.0061
	Test	0.3442	0.0070	0.0156	0.3481
Japan	**Train**	1.0053	0.0020	0.0025	0.9696
	Test	0.3413	0.0036	0.0032	0.3156
China	**Train**	1.0151	0.0020	0.0025	0.9956
	Test	0.3332	0.0078	0.0038	0.3361
New York	**Train**	1.0579	0.0097	0.0026	1.0057
	Test	0.3376	0.0244	0.0049	0.2977
S&P 500	**Train**	1.0466	0.0021	0.0024	1.0062
	Test	0.3303	0.0058	0.0028	0.3377
NASDAQ	**Train**	1.0788	0.0022	0.0029	0.9832
	Test	0.3615	0.0030	0.0038	0.3159

CHAPTER 4

CONCLUSION AND FUTURE WORK

In this study, machine learning algorithms and their hybrid performances are evaluated on nine different stock market index log-return time series data. Within this study, it becomes efficient to analyse the market and take relevant action.

The motivation of this thesis is to find an efficient way for predicting the stock market index. Since it is known that the importance of investing the real instrument, in this case, the stock market index, it is essential to follow the technological and economic developments and make logically accurate and reliable analyses. This study's roadmap has been; researching the most used and the most efficient algorithms, trying these algorithms with different datasets in a computational environment, and evaluating the process.

Considering previous studies, it is intended to use machine learning models in financial time series data and reach the model that gives the best result. Stock indices, which reflect the market and provide a broad perspective, are deemed appropriate for this study. Besides choosing LSTM and XGBoost, the most preferred and studied machine learning methods, combinations of these two models are tested, and their hybrid performances are evaluated. A total of nine stock market indexes were selected, three from European markets, three from Asian and three from American markets for this assessment, and although the sizes of the dataset are different from each other, they all are recorded with daily frequency. The size of dataset effects the computation time of the model. Moreover, if there are more data, then model can learn details more specifically, so its predictions become more accurate. The Germany stock mar-

ket index log-return data which has the minimum size of data could be an example of the effect of size on learning capacity. Therefore, four methods are implemented for each data: LSTM applied to log-return data, XGBoost applied to the residuals of LSTM, XGBoost applied to log-return data, and LSTM applied to the residuals of XGBoost. After all, implementations are carried out, error measures are evaluated, and comparisons are made.

To sum up, according to MASE values, for seven out of nine (%77.77) data, hybrid models give better results. It is observed that applying XGBoost to residuals of LSTM makes results more accurate. Only two data yielded better results using a single XGBoost model. As with all algorithms, not only XGBoost but also LSTM have both advantages and disadvantages. The average MASE value of XGBoost applied to residuals of LSTM for training is 0.03011 (min=0.00201 and max=0.00973) and for testing is 0.1414 (min=0.0030 and max=0.079). For computation time aspect, residual calculations last longer, but MASE values of XGBoost are close to MASE values of XGBoost applied to residuals of LSTM. Therefore, only XGBoost application could be considered according to the dataset and research.

The crucial issue is carefully selecting the accurate model with the right hyper-parameters for each data because there is no algorithm that yields the most superior model with the default hyper-parameters for all datasets.

Future studies will inevitably develop algorithms that will provide much more relevant and reliable results in a shorter computational time in this rapidly changing world with technology development.

REFERENCES

[1] Lstm layer, https://keras.io/api/layers/recurrent_layers/lstm/.

[2] Xgboost documentation-xgboost parameters, https://xgboost.readthedocs.io/en/latest/parameter.html.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning, 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283, 2016.

[4] M. Aditi, Understanding RNN and LSTM, 2019, https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

[5] R. Agrawal and R. Srikant, *Privacy-Preserving Data Mining*, Association for Computing Machinery, New York, NY, USA, 2000, ISBN 1581132174.

[6] W. Bao, J. Yue, and Y. Rao, A deep learning framework for financial time series using stacked autoencoders and long-short term memory, PLoS ONE, 12(7), 2017.

[7] W. Bao, J. Yue, and Y. Rao, A deep learning framework for financial time series using stacked autoencoders and long-short term memory, PLOS ONE, 12(7), pp. 1–24, 07 2017.

[8] A. G. Barnston, Correspondence among the correlation, rmse, and heidke forecast verification measures; refinement of the heidke score, Wea. Forecasting, 7(4), pp. 699–709, 1992.

[9] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, Advances in optimizing recurrent networks, IEEE International Conference on Acoustics, 8264-8, 2012.

[10] N. Bhargava, G. Sharma, R. Bhargava, and M. Mathuria, Decision tree analysis on j48 algorithm for data mining, 2013.

[11] P. Bühlmann, Bagging, boosting and ensemble methods, Handbook of Computational Statistics, 01 2012.

[12] P. Bühlmann and B. Yu, Analyzing bagging, Annals of Statistics, 30, 08 2002.

[13] K. Bjoern, V. Bruce, and F. Gavin, Financial time series forecasting with machine learning techniques: A survey, From the Selected Works of Bjoern Krollner, 2010.

[14] S. Borra and A. Di Ciaccio, Improving nonparametric regression methods by bagging and boosting, Computational Statistics and Data Analysis, 38, pp. 407–420, 02 2002.

[15] L. Breiman, Bagging predictors, Machine Learning, 24, pp. 123–140, 1996.

[16] L. Breiman, Out-of-bag estimation, 1996.

[17] L. Breiman, Random forests, Machine Learning, 45, pp. 5–32, 2001.

[18] J. Brownlee, Time series prediction with LSTM recurrent neural networks in python with keras, deep learning for time series, 2020, `https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras`.

[19] H. Bruno Miranda, S. Vinicius Amorim, and K. Herbert, Literature review: Machine learning techniques applied to financial market prediction, Expert Systems With Applications, 124, pp. 226–251, 2019.

[20] A. Buja and W. Stuetzle, Observations on bagging, Statistica Sinica, 16, pp. 323–351, 04 2006.

[21] M. Butler and D. Kazakov, The effects of variable stationarity in a financial time-series on artificial neural networks, in *2011 IEEE Symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr)*, pp. 1–8, 2011.

[22] Z. Chang, Y. Zhang, and W. Chen, Electricity price prediction based on hybrid model of adam optimized LSTM neural network and wavelet transform, Energy, 187, 2019.

[23] T. Chen and C. Guestrin, Xgboost: A scalable tree boosting system, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785,794, 2016.

[24] F. Chollet et al., Keras, 2015, `https://keras.io`.

[25] J. D. Cryer and K.-S. Chan, *Time Series Analysis With Applications in R*, Springer, 2008.

[26] A. Cutler, D. Cutler, and J. Stevens, *Random Forests*, volume 45, pp. 157–176, 01 2011, ISBN 978-1-4419-9325-0.

[27] J. Elman, Finding structure in time, Cognitive Science, 14(2), pp. 179–211, 1990.

[28] E. F. Fama, Random walks in stock market prices, Financial Analysts Journal, 51(1), pp. 75–80, 1995.

[29] Y. Freund, Boosting a weak learning algorithm by majority to be published in information and computation, 1995.

[30] Y. Freund and R. Schapire, Experiments with a new boosting algorithm, in *ICML*, 1996.

[31] J. Friedman, Greedy function approximation: A gradient boosting machine, Annals of Statistics, 29, pp. 1189–1232, 10 2001.

[32] J. Friedman, Stochastic gradient boosting, Computational Statistics and Data Analysis, 38, pp. 367–378, 02 2002.

[33] J. Friedman, T. Hastie, and R. Tibshirani, Additive logistic regression: A statistical view of boosting, The Annals of Statistics, 28, pp. 337–407, 04 2000.

[34] J. Friedman and B. Popescu, Importance sampled learning ensembles, 2003.

[35] C. W. J. Granger and T. Terasvirta, Modelling non-linear economic relationships, OUP Catalogue, Oxford University Press, (9780198773207), 1993.

[36] D. Grattarola and C. Alippi, Graph neural networks in tensorflow and keras with spektral, 2020.

[37] A. Graves, Sequence transduction with recurrent neural networks, Computer Science, 58(3), pp. 235–42, 2012.

[38] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 2012, ISBN 978-0-12-381479-1.

[39] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural Computation, 9(8), pp. 1735–1780, 1997.

[40] E. Holmes, M. Scheuerell, and E. Ward, *Applied time series analysis for fisheries and environmental data*, NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112, 2021.

[41] R. J. Hyndman, Another look at forecast accuracy metrics for intermittent demand, Foresight: The International Journal of Applied Forecasting, (4), pp. 43–46, 2006.

[42] W. Ju-Jie, W. Jian-Zhou, Z. Zhe-George, and G. Shu-Po, Stock index forecasting based on a hybrid model, Omega, 40, pp. 758–766, 2012.

[43] M. Jun, C. Jack C.P., X. Zherui, C. Keyu, L. Changqing, and J. Feifeng, Identification of the most influential areas for air pollution control using XGBoost and grid importance rank, Journal of Cleaner Production, 274, p. 122835, 2020, ISSN 0959-6526.

[44] I. Kandel and M. Castelli, The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset, ICT Express, 2020.

[45] T. Y. Kim, K. J. Oh, C. Kim, and J. D. Do, Artificial neural networks for non-stationary time series, Neurocomputing, 61, pp. 439 – 447, 2004, ISSN 0925-2312, hybrid Neurocomputing: Selected Papers from the 2nd International Conference on Hybrid Intelligent Systems.

[46] D. P. Kingma and J. L. Ba, Adam: A method for stochastic optimization, 2014.

[47] G. Krishnadas, Data-driven modelling for demand response from large consumer energy assets, 2018.

[48] M. Kuhn, The caret package, 2011, https://topepo.github.io/caret/index.html.

[49] M. Kumar and M. Thenmozhi, Forecasting stock index returns using ARIMA-SVM, ARIMA-ANN, and ARIMA-random forest hybrid models, International Journal of Banking, Accounting and Finance, 5(3), pp. 284–308, 2014.

[50] M. Kursa and W. Rudnicki, Feature selection with boruta package, Journal of Statistical Software, 36, pp. 1–13, 09 2010.

[51] M. B. Kursa, Boruta for those in a hurry, 2018, https://rdrr.io/cran/Boruta/f/inst/doc/inahurry.pdf.

[52] B. Labiad, A. Berrado, and L. Benabbou, Machine learning techniques for short term stock movements classification for moroccan stock exchange, pp. 1–6, 2016.

[53] A. Liaw and M. Wiener, Classification and regression by randomforest, Forest, 23, 11 2001.

[54] B. Luis, Overview of neural networks, 2017, https://medium.com/machinevision/overview-of-neural-networks-b86ce02ea3d1.

[55] J. Ma, J. C. Cheng, Z. Xu, K. Chen, C. Lin, and F. Jiang, Identification of the most influential areas for air pollution control using xgboost and grid importance rank, Journal of Cleaner Production, 274, 2020.

[56] S. Makridakis, A. Andersen, R. Carbone, R. Fields, M. Hibon, R. Lewandowski, J. Newton, E. Parzen, and R. Winkler, The accuracy of extrapolation (time series) methods: Results of a forecasting competition, Journal of Forecasting, 1, pp. 111–153, 1982.

[57] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, The M4 competition: 100,000 time series and 61 forecasting methods, International Journal of Forecasting, 36(1), pp. 54 – 74, 2020, ISSN 0169-2070.

[58] G. Mesnil, X. He, L. Deng, and Y. Bengio, Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding, Interspeech, 2013.

[59] K. Miao, F. Chen, and Z.-g. Zhao, Stock price forecast based on bacterial colony rbf neural network, Journal of Qingdao University(Natural Science Edition).

[60] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpu, Recurrent neural network based language model, INTERSPEECH 2010.

[61] M. Mohammed, M. Khan, and E. Bashier, *Machine Learning: Algorithms and Applications*, 07 2016, ISBN 9781498705387.

[62] H. Nassif, Learning sentiment and semantic relatedness in user generated content using neural models, 2016.

[63] M. Nykter and J. Kesseli, Classification of lymph node metastases in breast cancer with features from tissue images using machine learning techniques, 2017.

[64] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval, IEEE/ACM Transactions on Audio, Speech, and Language Processing, 24(4), pp. 694–707, 2016.

[65] H. Palangi, L. Deng, and R. Ward, Distributed compressive sensing: A deeplearning approach, IEEE Transactions on Signal Processing, 64(17), 2016.

[66] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, Predicting stock market index using fusion of machine learning techniques, Expert Systems with Applications, 42, 10 2014.

[67] L. Pauly, H. Peel, S. Luo, D. Hogg, and R. Fuentes, Deeper networks for pavement crack detection, 07 2017.

[68] K. Pawar, R. Jalem, and V. Tiwari, Stock market price prediction using LSTM RNN, Advances in Intelligent Systems and Computing book series: Emerging Trends in Expert Applications and Security.

[69] J. Quinlan, Induction of decision trees, Machine Learning, 1, pp. 81–106, 1986.

[70] A. M. Rather, A. Agarwal, and V. Sastry, Recurrent neural network and a hybrid model for prediction of stock returns, Expert Systems with Applications, 42, pp. 3234–3241, 2015.

[71] S. Remya and R. Sasikala, Performance evaluation of optimized and adaptive neuro fuzzy inference system for predictive modeling in agriculture, Computers and Electrical Engineering, 86, 2020.

[72] A. Robinson, An application of recurrent nets to phone probability estimation, IEEE Transactions on Neural Networks, 5(2), pp. 298–305, 1994.

[73] L. Rokach and O. Maimon, *Data mining with decision trees. Theory and applications*, volume 69, 01 2008.

[74] J. S. S. Hochreiter, Long short-term memory, Neural Competition, 9(8), pp. 1735–80, 1997.

[75] Q. Saad, *Realization and Optimization of Deep Learning Algorithm for Object Detection and Classification on FPGA*, Ph.D. thesis, 12 2016.

[76] S. Saed, Artificial neural network, 2019, `https://www.saedsayad.com/artificial_neural_network.htm`.

[77] R. Schapire, The strength of weak learnability, Machine Learning, 5, pp. 197–227, 2005.

[78] O. Sezer, U. Gudelek, and M. Ozbayoglu, Financial time series forecasting with deep learning : A systematic literature review: 2005-2019, 11 2019.

[79] G. Silahtaroğlu, *Veri Madenciliği Kavram ve Algoritmaları*, 2008.

[80] M. Spyros, S. Evangelos, and A. Vassilios, The M4 competition: 100,000 time series and 61 forecasting methods, International Journal of Forecasting, 36, pp. 54–74, 2020.

[81] K. Sungil and K. Heeyoung, A new metric of absolute percentage error for intermittent demand forecasts, International Journal of Forecasting, 32, pp. 669–679, 2016.

[82] F. E. H. Tay and L. Cao, Application of support vector machines in financial time-series forecasting, Omega, 29(4), pp. 309–317, 2001.

[83] A. M. Turing, Computing machinery and intelligence, Oxford University Press on behalf of the Mind Association, 59, pp. 433–460, 1950.

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

[85] D. Yan, Q. Zhou, J. Wang, and N. Zhang, Bayesian regularisation neural network based on artificial intelligence optimisation, International Journal of Production Research, 55(8), pp. 2266–2287, 2017.

[86] V. Yüncü and U. Fidan, Utilizing decision trees on employee decision-making processes: A model proposal, 12 2020.

[87] C. Yozgatlıgil, Lecture notes in stat 497–applied time series analysis, 2019.

[88] G. P. Zhang, B. E. Patuwo, and M. Y. Hu, A simulation study of artificial neural networks for nonlinear time-series forecasting, Computers and Operations Research, 28(4), 2001.

[89] N. Zhang, A. Lin, and P. Shang, Multidimensional k-nearest neighbor model based on EEMD for financial time series forecasting, Physica A: Statistical Mechanics and its Applications, 477(C), pp. 161–173, 2017.

[90] P. Zhang, Time series forecasting using a hybrid ARIMA and neural network model, Neurocomputing, 50, pp. 159–175, 2003.

[91] Y. Zhang, B. Yan, and M. Aasma, A novel deep learning framework: Prediction and analysis of financial time series using ceemd and lstm, Expert Systems with Applications, 159, pp. 1735–1780, 2020.

[92] H. Zheng, J. Yuan, and L. Chen, Short-term load forecasting using EMD-LSTM neural networks with a XGBoost algorithm for feature importance evaluation, Energies, 10(8), 2017.