Handelshøyskolen BI

GRA 19703 Master Thesis

Thesis Master of Science 100% - W

Predefinert informasjon

Startdato:	09-01-2023 09:00 CET	**Termin:**	202310
Sluttdato:	03-07-2023 12:00 CEST	**Vurderingsform:**	Norsk 6-trinns skala (A-F)
Eksamensform:	T		
Flowkode:	202310||11184||IN00||W||T		
Intern sensor:	(Anonymisert)		

Deltaker

Navn:	Johannes Øvrebø Haugland

Informasjon fra deltaker

Tittel *:	A comparative study of machine learning models in stock price prediction
Navn på veileder *:	John Chandler Johnson

Inneholder besvarelsen konfidensielt materiale?:	Nei	**Kan besvarelsen offentliggjøres?:**	Ja

Gruppe

Gruppenavn:	(Anonymisert)
Gruppenummer:	341
Andre medlemmer i gruppen:	Deltakeren har innlevert i en enkeltmannsgruppe

BI Norwegian Business School
Oslo, Spring 2023

A comparative study of machine learning models in stock price prediction

Johannes Haugland

Supervisor: John Chandler Johnson

Master thesis, Business Analytics

BI NORWEGIAN BUSINESS SCHOOL

Acknowledgements

This thesis represents the completion of my two-year Master program in Business Analytics at BI Norwegian Business School in Oslo. I would like to thank all those who have supported me along the way.

First and foremost, I would like to express my sincere thanks to my supervisor, Associate Professor John Chandler Johnson. His expertise and support have been important in defining the topic for the thesis and conducting the research in an analytical manner. I would also like to thank the faculty members at BI for their dedication to research and their role in broadening my knowledge and understanding of the field of business analytics.

I am very grateful that I have had the opportunity to work on this thesis, as this experience has extended my knowledge and skills in the field of machine learning and its applications in business analytics. The process of conducting this thesis has allowed me to acquire in-depth knowledge, valuable insight, and practical expertise on predictive modeling. Overall, I am grateful for this experience and personal growth.

Abstract

This thesis predicts one-day ahead adjusted closing price estimates of four different stocks listed at the Oslo Stock Exchange. The prediction uses a rolling window approach, utilizing 60 days of historical adjusted closing prices as input for each observation. The thesis conducts a comparative analysis, evaluating the prediction performance of five different models: Ordinary Least Squares, Support Vector Regression, Random Forest, Extreme Gradient Boosting, and Long Short-Term Memory networks.

The findings of the comparative analysis indicate that the Long Short-Term Memory (LSTM) network consistently outperforms the other models for all the stocks considered. These findings align with previous research that highlights the effectiveness of LSTM models in stock market prediction. While the thesis does not provide a definitive conclusion on market efficiency or inefficiency, the predictive performance of the LSTM model suggests the presence of potential inefficiencies in the market. Additionally, this thesis identifies key hyperparameters used to avoid overfitting as well as optimizing predictive performance for the LSTM model.

Overall, this research contributes to the existing literature by investigating the effectiveness of the models on a different market, the Norwegian stock market, as well as noting the importance of model specific hyperparameter tuning.

Table of Contents

List of figures

List of tables

1.0 Introduction

Predicting stock prices has always been a challenging task, often attempted using statistical tools and various prediction methods. The stock market is notorious for its unpredictable nature, constant fluctuations, and absence of a simple pattern. Achieving accurate stock price predictions is difficult due to numerous influencing factors, such as political events, global economic conditions, unforeseen incidents, and a company's performance. Although each individual stock is subject to unique variation, there are also similarities in how different stocks develop, including market trends, sector and industry influences, and investor sentiment.

While accounting for all these factors is challenging, it is worth noting that stock prices themselves reflect the impact of these factors, potentially containing valuable patterns and insights. Machine learning techniques have gained considerable attention over the last years as powerful tools to analyze and interpret such patterns. Although accurately predicting stock prices, especially based solely on price data, remains challenging, we can still learn from emerging patterns and leverage this information to make more informed investment decisions.

The existing body of literature on stock price prediction and machine learning models is extensive. However, it is important to recognize that different stock markets are influenced by various factors, which can lead to variations in the performance of machine learning models. Each model may excel in different tasks based on the specific characteristics of the market under consideration. While there are studies available that compare machine learning models, there is a notable gap in the literature when it comes to testing these models across different markets. Additionally, existing literature on stock price prediction and machine learning models often overlooks the crucial aspects of the hyperparameter tuning tailored to specific problems.

1.1 Research Question

This thesis aims to conduct a comparative analysis of five distinct machine learning models: OLS (Ordinary Least Squares), SVR (Support Vector Regression), Random Forest, XGBoost, and LSTM (Long Short-Term Memory). The primary objective is to create a model with the most accurate performance in predicting the adjusted closing price across four stocks that represent different industries within the Norwegian stock market. We use the adjusted closing price because it provides a more accurate depiction of the stock's true value over time. Through the evaluation and comparison of these models, the research aims of this thesis is to identify the most suitable model for analyzing complex patterns of the Norwegian stock market. While answering this question this thesis also addresses the importance of problem specific hyperparameter tuning and will also discuss the following questions:

- Is the market truly efficient or is there inefficiencies in the market?
- Are there any interesting patterns that emerge in the different stocks analyzed?

1.2 Literature review

Stock price prediction has been a subject of interest for a considerable period, where various traditional statistical methods have been employed to forecast stock prices. While these approaches may have originated many years ago, many of them continue to be utilized today in stock market analysis and prediction. However, with more complex algorithms being developed over the years, including more powerful computers, there has been an increase in the literature on utilizing advanced machine learning algorithms for this purpose. This increased interest can be attributed to the potential of machine learning in capturing complex patterns and improving prediction accuracy. Despite the growing body of research, there remains a lack of literature specifically addressing stock price prediction for the Norwegian stock market, as well as the significance of hyperparameter tuning in this context.

One of the earliest attempts to apply machine learning techniques to stock prediction was performed by Galler and Kryzanowski (1993). In their study, they explored the use of an artificial neural network (ANN) to predict stock returns based on a range of

financial and economic variables. The findings revealed promising results, as the ANN achieved a significant level of accuracy in predicting stock returns.

More recently, Kumar et al. (2018) performed an extensive comparative analysis of machine learning models for stock market trend prediction. Their study aimed to address the challenges of predicting stock prices by applying various machine learning techniques, including Support Vector Machine (SVM), Random Forest, K-Nearest Neighbor (KNN), Naive Bayes, and Softmax. Their findings indicated that the Random Forest algorithm demonstrated superior performance when dealing with large datasets, suggesting its suitability for handling complex market dynamics.

Moreover, Nabipour et al. (2020) performed an investigation on the applicability of machine learning models for stock market prediction using data from the Tehran Stock Exchange. Their study focused on four specific stock market groups and compared the performance of nine machine learning models, including Decision Tree, Random Forest, Adaboost, XGBoost, SVC, Naïve Bayes, KNN, Logistic Regression, and Artificial Neural Network (ANN). The findings highlighted the effectiveness of deep learning methods, specifically Recurrent Neural Network (RNN) and Long short-term memory (LSTM), in predicting stock market trends.

On the topic of importance of hyperparameter tuning for stock price prediction, Yadav et al. (2020) Conducted a study on the optimization of LSTM models for time series prediction of the Indian stock market. They concluded that the performance of LSTM models is highly dependent on the choice of hyper-parameters which needs to be carefully considered.

1.3 Thesis structure

This thesis will begin by explaining the relevant financial and technical theories related to the topic. Chapter 3 will describe the methodology used, including how the data is organized, the modeling approach, the specific models used, and how they are evaluated. Chapter 4 presents the results obtained from the models. Chapter 5 discusses and analyzes the findings from the results, and finally, Chapter 6 will present the conclusion of the thesis.

2.0 Theory

I will in this chapter discuss various theories utilized in the research. I will start by examining the efficient market hypothesis, followed by an exploration of the technical theories related to machine learning.

2.1 Theoretical Foundations of Stock Prediction

2.1.1 The Efficient Market Hypothesis (EMH)

The random walk theory, proposed by Louis Bachelier in "Théorie de la spéculation," offers a foundational concept for comprehending market dynamics. Initially developed for games of chance, this theory has found extensive application in financial markets. A random walk represents a mathematical model that captures the path of a variable through independent and unpredictable steps, without considering any relationship to past or future steps (Levy, 1957).

Financial theory posits that a random walk implies that asset price changes are random, rendering past prices ineffective in predicting future movements. Furthermore, it suggests that the stock market is efficient, reflecting all available information, which aligns with the assumptions of the efficient market hypothesis.

The random walk theory challenges the idea that traders can consistently profit from timing the market or exploiting stock price trends and patterns through technical analysis. However, critics argue that the theory oversimplifies the intricate nature of financial markets by disregarding the influence of market participants, their behavior, and their actions on prices and outcomes.

The prominence of the random walk concept grew in the 1960s through the research of economists like Burton Malkiel and Eugene Fama, who aimed to explain stock price behavior. According to the random walk hypothesis, stock prices lack discernible patterns or trends and, consequently, cannot be predicted based on past prices or other available information.

This concept closely connects with the efficient market hypothesis (EMH), asserting that financial markets are efficient and asset prices fully incorporate all available information. The random walk hypothesis forms the basis for the weak form of EMH, which argues that past price movements and trading volume data are already factored into stock prices and, thus, cannot predict future price movements (Smith, 2023).

The efficient market hypothesis (EMH), introduced by Eugene Fama in 1970, is a fundamental theory in finance that suggests financial markets are efficient, meaning that the prices of assets traded in the market reflect all publicly available information. The theory has been the subject of much debate and scrutiny since its inception, with proponents arguing for its validity and critics raising objections based on observed market anomalies and behavioral biases (Naseer & Tariq, 2015; Becker, 2021). Despite the ongoing discourse, the EMH remains a widely studied and influential theory in finance, providing a framework for understanding market behavior and informing investment strategies.

Fama distinguishes between three forms of the EMH: weak form, semi-strong form, and strong form. Each form represents a different level of market efficiency and the extent to which different types of information are reflected in asset prices (Naseer & Tariq, 2015).

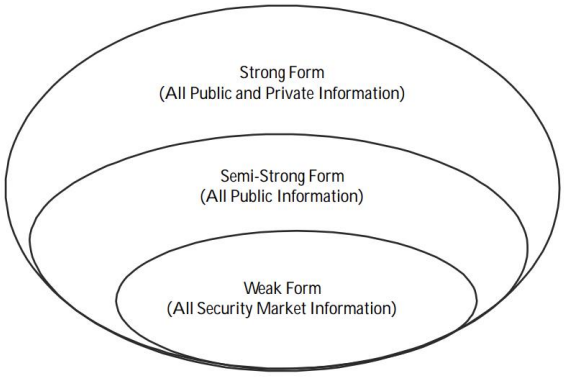

Figure 1 The different forms of EMH (adapted from Naseer & Tariq, 2015).

1. *Weak Form:* The weak form of the EMH states that current stock prices fully reflect all historical prices and trading volume data. This implies that it is not

possible to predict future stock prices based solely on analyzing past price data. According to the weak form of EMH, all security market information regarding a company is already incorporated into the stock price, and any attempts to generate abnormal returns using this information will be unsuccessful. Therefore, the weak form EMH suggests that technical analysis, which involves analyzing past price and volume data, is not useful in making investment decisions because it cannot identify patterns that will enable investors to earn excess returns.

2. *Semi-strong form:* The semi-strong form of the EMH states that current stock prices fully reflect all publicly available information. This includes publicly available data such as financial statements, news releases, macroeconomic data, and market speculations. The semi-strong form of EMH assumes that all information that is publicly accessible is promptly and properly reflected in the stock price, eliminating any opportunity for investors to profit excessively from trading on this information. Therefore, the semi-strong form of EMH suggests that neither technical analysis nor fundamental analysis can be used to consistently generate abnormal returns since all publicly available information is already reflected in the stock price.

3. *Strong form:* The strong form of the EMH states that current stock prices fully reflect all public and private information. This means that not only is all publicly available information incorporated into the stock price, but also all private or insider information known to a selected group of people, such as company executives or major shareholders. According to the strong form of EMH, even information that is known only to insiders is already reflected in the stock price, and any attempts to generate abnormal returns using this information will be unsuccessful. Therefore, the strong form of EMH suggests that no investor, regardless of their level of expertise or access to information, can consistently earn abnormal returns.

2.1.1.2 Criticism

While acknowledging the existence of market efficiency to some extent, it is widely recognized that certain pockets of inefficiency, termed market anomalies, persist within the broader concept. Notable examples of these anomalies include momentum effects and value investing strategies.

Momentum trading involves investors buying stocks that have demonstrated strong performance over a specific period (winner stocks) while simultaneously short selling stocks that have performed poorly during the same period (loser stocks). Empirical evidence indicates that this zero-investment strategy can generate a positive annual return of up to 12 percent (Singal, 2003).

Value investing is an investment strategy focused on identifying undervalued stocks or assets. It operates on the belief that the market occasionally misprices stocks, resulting in discrepancies between a company's intrinsic value and its market price. Value investors aim to capitalize on these pricing inefficiencies by purchasing undervalued stocks with the expectation of future recognition and price appreciation.

Behavioral bias represents another pocket of inefficiency. Investors' irrational behaviors, such as reluctance to realize losses, prematurely taking profits, insufficient diversification, and cognitive biases, contribute to mispricings in financial markets. These mispricings persist as investors may fail to adjust their behavior even when new information emerges. For instance, loss aversion may lead investors to hold onto declining assets, causing prices to deviate from their true value. Selling winning investments prematurely and neglecting diversification can also result in mispriced assets. Cognitive biases and selective perception further contribute to the persistence of mispricings (Singal, 2003).

These findings suggest the existence of exploitable patterns or trends in stock prices that enable investors to earn abnormal profits, contradicting the hypothesis of complete market efficiency.

It is important to note that the presence of market anomalies does not negate the overall efficiency of financial markets. Instead, it underscores markets' intricate dynamics and opportunities for investors to exploit mispricings and inefficiencies. Ongoing research continues to analyze these anomalies, aiming to better understand their underlying causes and their implications for investment strategies.

2.2.1.2 Empirical Evidence

The efficient market hypothesis is a widely debated concept in finance. Empirical evidence plays a critical role in evaluating the validity of this hypothesis. In this section, I present a summary of some of the empirical evidence regarding the EMH, dividing it into two categories: evidence supporting the EMH and evidence contradicting the EMH. By examining these empirical findings, we can gain insights into the efficiency of financial markets and its implications.

Empirical evidence supporting EMH

- Fama, Fisher, Jensen, and Roll (1969) found that the largest positive abnormal returns occur in the first 3-4 months after split events, supporting gradual price adjustments in capital markets.
- Konak and Seker (2014) analyzed the FTSE 100 index and found that it adhered to the random walk theory, supporting the weak form of the efficient market hypothesis.
- Malkiel (2003) argued that anomalies in stock prices do not create trading opportunities for earning extraordinary risk-adjusted returns, indicating that capital markets are efficient and less predictable.

Empirical evidence contradicting EMH

- Ball and Brown (1968) observed that stock prices react slowly to accounting income announcements, contradicting the idea of instantaneous price adjustments.
- Bernard and Thomas (1990) found high autocorrelation of stock prices for the first 3 lags of the regression, suggesting a slow adjustment process and market inefficiencies.

- Studies by Chowdhury, Howe, and Lin (1993) as well as Pettit and Vanketash (1995) revealed that insiders consistently achieved significant abnormal returns, challenging the notion of market efficiency.

2.2 Machine Learning

Machine learning is a field of study focused on developing computer algorithms that can learn from data without explicit programming. It encompasses various techniques, with two main branches being supervised and unsupervised learning, each offering distinct advantages and limitations. Machine learning is widely applied in numerous consumer products and features, including spam filters, recommendation systems, and, pertinent to this thesis, stock prediction. Machine learning algorithms range from rule-based systems like algorithmic trading to sophisticated deep learning networks that learn from their own errors and enhance their performance over time (Géron, 2019).

I will delve into different types of machine-learning techniques relevant to the modelling framework that will be utilized in the subsequent analyses.

2.2.1 Supervised Learning

In supervised learning, we work with a dataset of labeled examples $\{(x_i, y_i)\}_{i=1}^{N}$ where each observation x_i among the population N is known as a feature vector. A feature vector represents an ordered list of numerical properties that describe the observed phenomena, with each dimension $j = 1, \dots, D$ containing a specific value denoted as $x^{(j)}$ (Burkov, 2019).

Models can learn and generate predictions using supervised learning, a potent machine learning technique, by observing input-output pairs. In the context of the financial market, supervised learning is essential for assessing and forecasting market trends, making investment decisions, and optimizing trading methods.

When learning under supervision, the model is given input data that can include a variety of financial metrics, such as stock prices, trade volumes, underlying economic variables, and technical indicators. A corresponding output label or target value describing the desired prediction or result is present for each input data point. In the case of stock predictions, the input could for example be past price data, and the output label could be the anticipated future price movement.

By analyzing the historical input-output pairs, the model aims to generalize patterns and relationships in the data, enabling it to make predictions on new, unseen data (Russell & Norvig, 2010).

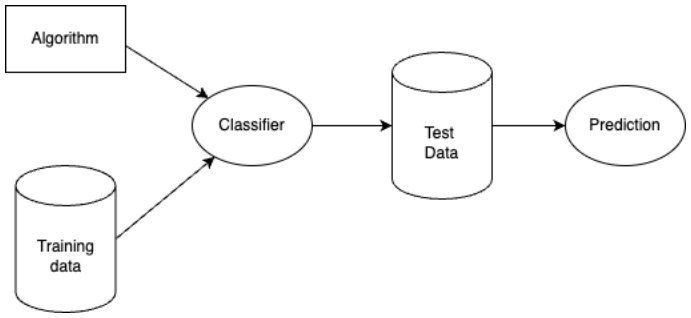

Figure 2 Logic of supervised learning

There are two fundamental tasks that arise based on the nature of the data or the approach taken by the model builder. These tasks are known as regression and classification.

2.2.1.1 Regression

Regression is a supervised learning task that involves predicting a real-valued label or target for an unlabeled example. It aims to establish a relationship between input variables and a continuous output variable. The goal is to create a regression model using labeled examples, which can then be used to predict the target value for new, unseen data points. The regression model learns patterns and relationships in the labeled data to make accurate predictions on unlabeled instances (Burkov, 2019).

2.2.1.2 Classification

Classification is a supervised learning task that involves automatically assigning a label to an unlabeled example. It is commonly used for tasks such as spam detection, sentiment analysis, and image recognition. In classification, a collection of labeled examples is used to train a classification learning algorithm, which then produces a model capable of predicting the label for new, unseen data points. The model's output can be a direct label or a numerical value, such as a probability, which can be interpreted by analysts to determine the appropriate label. Classification can be binary, with two classes, or multiclass, with three or more classes (Burkov, 2019). The prediction of stocks could be formulated as a classification problem by defining if a stock movement is positive or negative.

2.2.2 Unsupervised Learning

Unsupervised learning involves analyzing unlabeled datasets to uncover patterns and structures. Unlike supervised learning, where the data is represented by a collection of labeled examples, the dataset in unsupervised learning consists of a collection of unlabeled examples $\{x_i\}_{i=1}^{N}$ where x is a feature vector. The goal is to create models that transform or derive insights from these feature vectors to solve practical problems. Unsupervised learning explores the data's inherent structure, statistical properties, and relationships to extract meaningful representations, identify similarities or anomalies, reduce dimensionality, and discover hidden patterns, enabling us to gain valuable insights from unstructured data without explicit labels (Burkov, 2019).

2.2.4 Neural Networks

Neural networks, a popular class of machine learning techniques, provide a versatile approach capable of approximating various types of target functions. Inspired by the learning mechanisms observed in biological organisms, neural networks leverage the fundamental role of neurons in the human nervous system. These neurons form interconnected networks through axons and dendrites, creating synapses with adaptable strengths that facilitate learning in living organisms. In artificial neural

networks, computational units called neurons emulate this biological mechanism, processing and exchanging information within the network.

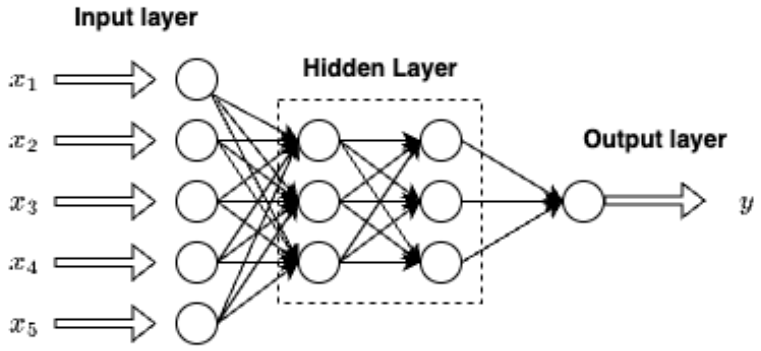

Figure 3 Basic architecture of feed-forward network with two hidden layers

Figure 3 depicts the fundamental structure of a feed-forward neural network, a type of multilayer neural network widely used in machine learning. This architecture encompasses multiple computational layers, including hidden layers positioned between the input and output layers. The network's input is processed and transformed in large part by the hidden layers, which were given their name because of the computations that are hidden from view. From the input layer to the output layer, information moves across the network in a single direction. Every node in a feed-forward network is often connected to nodes in the layer below it, determining the network's interconnectivity. The determination of the number of layers and the configuration of nodes in each layer significantly shape the neural network's structure, influencing its capacity to learn and make predictions (Aggarwal, 2018).

2.2.4.1 Neurons

In the context of artificial neural networks, neurons serve as the fundamental units responsible for computing the network's output based on the given inputs. The network propagates the computed values from the input neurons to the output neuron(s), employing intermediate weights as adjustable parameters. Learning takes place through the modification of these weights that connect the neurons. Similar to the role of external stimuli in learning observed in biological organisms, artificial neural networks rely on training data comprising input-output pairs to provide the

necessary stimulus for learning the desired function. By analyzing these examples, the network adapts its internal parameters to approximate the underlying relationships and enhance its predictive capabilities.

Figure xx (Illustration of a neuron)

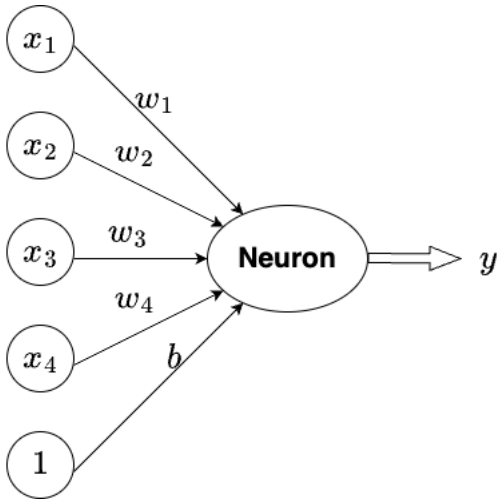

Figure 4 Illustration of a neuron

Figure 4 illustrates a single neuron within a neural network. The neuron receives inputs x and assigns weights w to each input. These weighted inputs are then aggregated, and a bias term b is introduced. The resulting value undergoes a non-linear transformation through the neuron's activation function, denoted as f, to compute the output of the neuron. Mathematically, the activation function is represented as $f(\sum(wx) + b)$ or as $f(\bar{X}\bar{W} + b)$. The choice of activation function plays a critical role in shaping the neuron's behavior and enabling the network to learn complex relationships and make accurate predictions (Aggarwal, 2018).

2.2.4.2 Activation function

Activation functions are important to the functioning of neural networks because they introduce nonlinearity to the network, which enables the network to capture complicated patterns in the data. The mathematical notation of the activation function can be denoted as Φ, and it is applied to the weighted input of a neuron, denoted as $(w \cdot x)$. As a result, a neuron performs two distinct functions: the summation of

inputs denoted by the Σ symbol and the activation function denoted by Φ. The pre-activation value refers to the computed value before applying the activation function, while the post-activation value represents the value obtained after the activation function is applied. Although the output of a neuron is always the post-activation value, pre-activation variables are used in various analyses such as the computations of the backpropagation algorithm.

The most fundamental activation function, $\Phi(\cdot)$, is the identity or linear activation, which lacks nonlinearity. It can be expressed as $\Phi(v) = v$, where v corresponds to the weighted input ($\bar{X}\bar{W} + b$). The linear activation provides a one-to-one mapping between the input and output without introducing any nonlinearity. It is often used in regression tasks or when the neural network needs to approximate a linear function. However, its limited expressive power restricts its effectiveness in modelling complex relationships (Aggarwal, 2018). There are several different activation functions, but some of the most common are:

Sign function

The sign function, denoted as

$$\Phi(v) = sign(v)$$

maps positive values to 1 and negative values to -1, effectively thresholding the output. It is mainly used in binary classification problems, where the goal is to separate instances into two classes.

Sigmoid function

The sigmoid activation function, also known as the logistic function, is defined as

$$\Phi(v) = \frac{1}{(1 + e^{-v})}$$

It maps the input to a range between 0 and 1, representing a probability-like value. Sigmoid functions are commonly used in binary classification problems, as they can

output a probability indicating the likelihood of an instance belonging to a particular class.

Tanh

The hyperbolic tangent (tanh) activation function, denoted as

$$\Phi(v) = \frac{(e^{2v} - 1)}{(e^{2v} + 1)}$$

maps the input to a range between -1 and 1. Like the sigmoid function, the tanh function is often used in classification problems but provides stronger nonlinearity than the sigmoid function.

ReLU

The Rectified Linear Unit (ReLU) activation function is defined as

$$\Phi(v) = \max\{v, 0\}$$

and is widely used in deep learning. It introduces nonlinearity by outputting the input directly if it is positive and 0 otherwise. ReLU is computationally efficient and helps alleviate the vanishing gradient problem, making it suitable for deep neural networks. It is often used in various tasks, including image recognition and natural language processing.

2.2.4.3 Layers

Input Layer

A neural network's input layer is the first layer to take in input data for processing. It functions as the link between the network's internal computations and external data. The dimensionality of the input data determines how many nodes there are in the input layer, where each node stands for a distinct feature or attribute. Instead of carrying out any calculations, the input layer acts as a conduit for the input values to be transmitted to the network's higher layers. Prior to producing an output, the values

from the input layer are propagated forward through the network where they are subjected to interactions and transformations in the hidden layers.

Hidden Layer

The hidden layer is a crucial component of neural networks and plays a significant role in processing and transforming the input data. Positioned between the input and output layers, the hidden layer consists of multiple neurons that collectively perform complex computations on the input. Unlike the input and output layers, the computations within the hidden layer are not directly visible or accessible to the user. The hidden layer serves as an intermediary, extracting and abstracting relevant features from the input data, enabling the network to learn intricate patterns and make accurate predictions. The number of hidden layers and the number of neurons in each hidden layer are key architectural decisions that heavily influence the network's capacity to model complex relationships. The hidden layer acts as a powerful information processing unit, transforming the input into a higher-dimensional representation that facilitates effective learning and generalization by the neural network.

Output Layer

The output layer is the final layer of a neural network and is responsible for producing the network's predictions or outputs. It receives inputs from the previous layers, which have been through a series of computations and transformations. The structure and functionality of the output layer depend on the nature of the problem. For example, in a classification task, the output layer typically consists of multiple neurons, with each neuron representing a distinct class and producing a probability score indicating the likelihood of the input belonging to that class. In regression tasks, the output layer may have a single neuron that directly outputs the predicted continuous value. The output layer serves as the final stage of the network's computation, providing the desired outputs based on the learned patterns and relationships in the input data (Russell & Norvig, 2010).

2.2.4.4 Weights

The weights in a neural network are fundamental components that play a crucial role in determining the network's behavior and performance. They serve as the learnable parameters that control the strength and importance of the connections between neurons. Each connection between neurons is associated with a weight, which represents the significance or contribution of the input from one neuron to the activation of the connected neuron.

During the training process, the weights are iteratively adjusted to optimize the network's performance and improve its ability to accurately model and predict the desired outputs. The initial weights are usually assigned randomly, and then updated using optimization algorithms. The goal is to find the set of weights that minimizes a defined loss or error function, effectively guiding the network towards better predictions (Russell & Norvig, 2010).

2.2.4.5 Learning Rate

The learning rate is a crucial hyperparameter in neural networks that determines the step size at which the weights are updated during the training process. It controls the rate at which the network learns and adjusts its parameters in response to the training data. The learning rate plays a significant role in the convergence speed and the quality of the final learned model.

A high learning rate may result in rapid weight updates, which can lead to overshooting the optimal solution and instability in the training process. On the other hand, a low learning rate may cause slow convergence and potentially get stuck in suboptimal solutions. Choosing an appropriate learning rate is essential to strike a balance between learning efficiency and convergence stability (Russell & Norvig, 2010).

2.2.3.1 Recurrent Neural Network (RNN)

A feed-forward neural network, commonly used for analyzing multidimensional data, assumes that the attributes are largely independent of each other. However, in certain

data types like time series data, there exist sequential dependencies among the attributes (Aggarwal, 2018). These dependencies pose a challenge for traditional feed-forward networks but can be effectively addressed by recurrent neural networks (RNNs).

Unlike feed-forward networks, RNNs are designed to handle sequential data by allowing cycles in the computation graph. These cycles introduce a delay, enabling units to take their own output from previous steps as input in the current computation. This unique property empowers RNNs with internal state or memory. The inputs received at earlier time steps influence the RNN's response to the current input, making them well-suited for analyzing sequential data (Russell & Norvig, 2010).

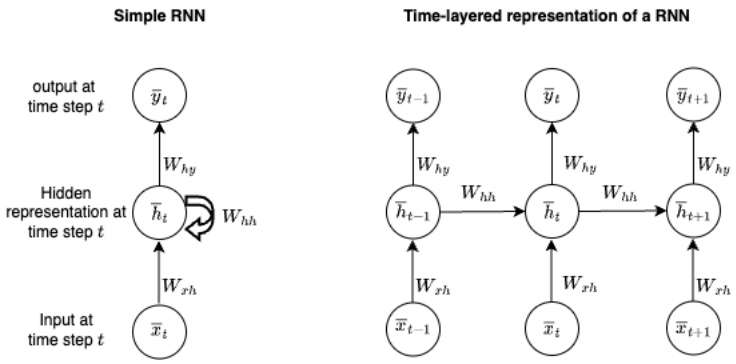

Figure 5 RNN architecture

Consider a time series dataset, such as stock prices, where the values at different timestamps are closely related. Treating each timestamp's value as an independent attribute would result in the loss of valuable information regarding the relationships among these timestamps. For instance, the value of a time series at time 't' is intricately connected to its values in the preceding window. However, this vital relationship is disregarded when individual timestamps are treated independently of each other.

3.0 Methodology

3.1 Data

The dataset used in this thesis consists of four stocks listed on the Oslo Stock Exchange Benchmark Index (OSEBX) from the period 01.01.2001-01.01.2023. OSEBX serves as the primary equity index of the Norwegian stock market. The selection of OSEBX as the target index was motivated by two primary considerations.

Firstly, this comparative study aims to identify the most suitable models for predicting the adjusted closing price of stocks within the Norwegian market. By focusing on constituents listed on OSEBX, we can gain valuable insights into the performance of different models specifically tailored to the unique characteristics of the Norwegian stock market. Secondly, OSEBX was chosen due to its comparatively lower liquidity compared to broader indices like the S&P 500. Market liquidity refers to the ease with which securities can be bought or sold without causing significant price changes (Scott, 2022). Selecting a less liquid index such as OSEBX allows us to explore features of a potentially less efficient market, which can lead to interesting findings in terms of market predictions.

The analysis includes the following stocks: Equinor, DNB, Telenor, and MOWI. The selection of these stocks was based on two main factors: market capitalization and industry. These stocks operate in different industries, offering an opportunity to investigate potential variations in model performance across diverse sectors. Furthermore, by choosing companies with high market capitalization, we ensure sufficient liquidity for accurate pattern identification, thus enhancing the precision of the models.

3.2 Feature engineering

Feature engineering is a crucial aspect of machine learning, involving the careful selection and transformation of relevant features from raw data to enhance the

performance of predictive models. In the context of this comparative thesis, where the objective is to identify patterns in the historical data of the Norwegian stock market while avoiding overfitting and data leakage, a focused approach will be employed. The models will be trained exclusively on the historical adjusted closing price, utilizing it as both the feature and target variables. The adjusted closing price refers to the stock's closing price that has been modified to account for factors such as dividends, stock splits, and other corporate actions (Ganti, 2020).

The adjusted closing price is particularly advantageous because it incorporates essential factors such as dividends, stock splits, and other adjustments, providing a more accurate depiction of the stock's true value over time. By concentrating solely on the adjusted closing price, the models can effectively capture the underlying trends and patterns without being influenced by potentially noisy or irrelevant features. This streamlined methodology significantly reduces the risk of overfitting and enables a robust analysis of the historical price data, facilitating meaningful comparisons among different stocks.

Compared to open and closing prices, the adjusted closing price offers notable benefits. Firstly, it ensures a more stable and consistent representation of a stock's value by accounting for various factors that can impact the stock price. This results in a smoother and more reliable signal for the models, enhancing their ability to identify meaningful patterns and make accurate predictions.

Furthermore, the utilization of the adjusted closing price helps mitigate bias and noise in the data. Open prices can be influenced by market sentiment and trading activities at the beginning of the trading session, while closing prices can be affected by news announcements or trading volume. By focusing on the adjusted closing price, these biases and external factors are minimized, leading to a more precise representation of the stock's intrinsic value.

To structure the feature as a target variable, the model uses 60 days of data to predict the target variable (the 61st day). The reasoning behind this approach is to capture

underlying dependencies in the long-term development in the price, which can be shown as the following visual representation:

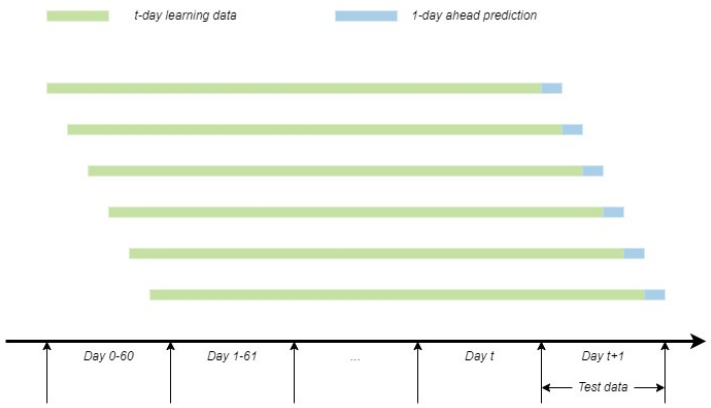

Figure 6 Illustration of a rolling window approach

In summary, the decision to use only the adjusted closing price as the feature and target variables in this thesis is driven by the aim to identify models that excel at uncovering patterns in the historical data of the Norwegian stock market. By doing so, the models can effectively capture the true value of the stocks over time, mitigate the risk of overfitting, and facilitate a comprehensive comparative analysis.

3.3 Research design

3.3.1 Train, test, and validation split

A well-structured approach to train, test, and validate is crucial for building a good machine learning model. The process involves dividing the available data into three distinct sets: training, validation, and test. Each set serves a specific purpose in evaluating and validating the model's performance.

The training set is used to train the model by feeding it with input data and corresponding target outputs. During training, the model learns to generalize patterns and make accurate predictions based on the provided examples. However, training alone is not sufficient to assess the model's true capabilities.

The validation set plays a critical role in fine-tuning the model's configuration, such as adjusting hyperparameters like the number of layers or layer sizes. By evaluating the model's performance on the validation data, we gain insights into how well it generalizes to unseen examples. This process is akin to learning from feedback, as the model's performance guides the selection of optimal hyperparameters. However, it is important to note that the model is never directly trained on the validation set.

Here comes the crucial point: repeatedly tuning the model based on its performance on the validation set can lead to overfitting. Overfitting occurs when the model becomes too specialized in capturing the nuances of the validation data, losing its ability to generalize to new, unseen data. This happens because information about the validation set "leaks" into the model during the tuning process. Although minimal leakage may not pose a significant problem, frequent tuning gradually incorporates more and more information from the validation set, which compromises its reliability.

In order to accurately assess the model's performance and its capacity to apply learned knowledge to new situations, it is essential to utilize a distinct dataset that has not been previously encountered, commonly referred to as the test dataset. The test set serves as an unbiased benchmark to assess the model's performance on completely new data. It is essential that the model has no prior access to any information from the test set, even indirectly. By evaluating the model on the test set, we gain a reliable estimate of its performance on real-world data.

In summary, a well-designed train, test, and validation split ensures a rigorous evaluation of a machine learning model. It prevents overfitting by keeping the tuning process separate from the test set, preserving the model's ability to generalize. By using distinct datasets for training, validation, and testing, we can build robust models that perform well on unseen data, ultimately improving their reliability and real-world applicability (Chollet, 2017).

3.3.2 Hyperparameters and Cross-Validation

Hyperparameters are essential components of machine learning models as they are predefined parameters that control model capacity and regularization. Setting these hyperparameters correctly is crucial to strike a balance between underfitting and overfitting, ensuring optimal model performance. However, finding the best hyperparameters is a challenging task. While we can learn them from the data, relying solely on the training data may lead to overfitting, as it tends to select maximum model capacity. To overcome this, a validation set, independent of the training process, is employed. In situations where the dataset is small, using a single validation set for evaluation may not yield reliable results. To tackle the issue, we can implement time series cross-validation (Joseph, 2022). The cross-validation procedure involves a set of test sets, where each test set comprises a single observation. The training set corresponding to each test set only includes observations that occurred prior to the observation in the test set. This ensures that no future observations are used in constructing the forecast. However, due to the limited data available in small training sets, it is not feasible to obtain reliable forecasts. As a result, the earliest observations are excluded from being considered as test sets in order to ensure the reliability and accuracy of the forecasting process (Hyndman & Athanasopoulos, 2018).

Cross-Validation

For the models used in this thesis, a time series cross-validation approach is implemented to evaluate the performance of the models. The data for different stocks is initially divided into two sets: the training set and the test set. The training set is used to train the models, while the test set is held out for evaluation purposes.

Within the time series cross-validation loop, the test set is further divided into five folds. Each fold is used as a validation set, and the remaining data from the test set, along with the training set, are used for model training. This ensures that the models are trained on historical data preceding the observations in the validation set, preventing any data leakage from future observations. The cross-validation approach used in this thesis is illustrated in figure 7.

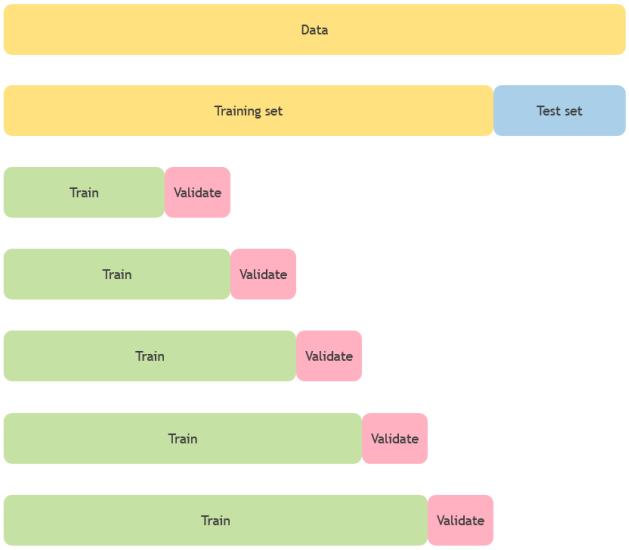

Figure 7 Fivefold cross-validation

Hyperparameters

Hyperparameters play a vital role in configuring and optimizing the performance of machine learning models within the experimental setup. Unlike the parameters that are learned from the data during training, hyperparameters are predefined settings that govern various aspects of the model's behavior and greatly influence its performance.

To determine the optimal values for hyperparameters, I have implemented GridSearch, a technique that exhaustively tests a predefined set of hyperparameter combinations. In each fold of the cross-validation loop, the model learns patterns based on the range of hyperparameters provided. It remembers these patterns and proceeds to the next fold, searching for additional patterns based on the updated knowledge. Once the model has traversed all the folds, it employs the best overall parameters to make predictions on the test set. This approach allows for systematic exploration and selection of hyperparameters that yield the best performance.

3.4 Models

This thesis compares five different models, Ordinary least squares, support vector regression, random forest, extreme gradient booster, and long short-term memory network. I will in this section present the different models, the model structures, why the models are used, fine tuning, and the specific evaluation metrics used.

3.4.1 Model Selection

3.4.1.1 Linear regression

For the linear regression model, I use the scikit-learn library's linear regression model, which fits the regression model based on the principles of ordinary least squares (OLS) linear regression.

Linear regression is a statistical technique that seeks to model the relationship between a dependent variable and a single independent variable. It assumes a linear association between the variables and aims to estimate the parameters that define this relationship. One commonly used method for fitting a linear regression model is Ordinary Least Squares (OLS). OLS involves minimizing the sum of the squared differences between the observed values of the dependent variable and the predicted values from the linear model (James et al., 2013).

While a simple linear regression model may not be the most suitable approach for predicting the adjusted closing price of stocks due to its inherent assumptions of linearity and independence, it still holds value as a benchmark model. As a benchmark, the linear regression model provides a baseline against which the performance of more advanced machine learning algorithms can be evaluated and compared.

Moreover, the linear regression benchmark allows us to evaluate the interpretability of more complex models. While advanced algorithms may yield higher predictive accuracy, they often sacrifice interpretability, making it challenging to understand the underlying drivers of the adjusted closing price movements.

3.4.1.2 Support Vector Regression

Support vector regression (SVR) is a machine learning algorithm that aims to estimate the relationship between a dependent variable and independent variables by minimizing the empirical risk, which is the sum of the loss function and a regularization term. The loss function determines the penalty for deviations between the predicted values and the actual values.

In SVR, a symmetrical tube of minimal radius is formed around the estimated function, and points falling within this tube are considered acceptable predictions. The tube is defined by a threshold value, often denoted as ε.

The mathematical representation of SVR can be formulated as follows:

Given a training dataset with n observations, denoted as (x_1, y_1), (x_2, y_2),…, (x_n, y_n), where x_i represents the independent variables and y_i represents the corresponding dependent variable, the objective is to find the optimal function $f(x)$ that minimizes the empirical risk.

The basic formulation of SVR involves finding a hyperplane that lies within the tube and maximizes the margin around it. The hyperplane is defined by the equation: $f(x) = w * x + b$.

Here, w represents the weight vector and b represents the bias or intercept term.

To incorporate the tube and handle deviations, SVR introduces a loss function that penalizes points outside the tube. One commonly used loss function is the ε -insensitive loss function, which is defined as: $L\big(y, f(x)\big) = \max\left(|y - f(x)| - \varepsilon, 0\right)$

In this equation, y represents the actual value, and $f(x)$ represents the predicted value by the SVR model. The loss function computes the deviation between the actual and predicted values, subtracts the threshold ε, and takes the maximum of this result and

zero. Points within the tube will have a loss of zero, while points outside the tube will have a positive loss.

The objective of SVR is to minimize the sum of the loss function over all training observations, subject to a regularization term to control the complexity of the model. The regularization term helps prevent overfitting by adding a penalty for large weight values.

The formulation of SVR can be written as an optimization problem:

minimize $^{1}/_{2}\,\|w\|^2 + C\sum L(y, f(x))$

subject to the constraints: $|y - f(x)| \leq \varepsilon$

Here, $\|w\|^2$ represents the squared Euclidean norm of the weight vector w, C is a regularization parameter that balances the trade-off between the margin and the loss function, and the sum is taken over all training observations.

Solving this optimization problem yields the optimal weight vector w, the bias term b, and the support vectors, which are the data points lying on the boundary of the tube.

By incorporating the appropriate kernel function, SVR can also handle nonlinear relationships between variables. The kernel function implicitly maps the data points into a higher-dimensional feature space, allowing the use of linear methods in this transformed space (Awad & Khanna, 2015).

In summary, SVR involves finding the optimal hyperplane that maximizes the margin while minimizing deviations outside the tube. The optimization problem incorporates the ε-insensitive loss function, a regularization term, and a kernel function to handle nonlinear relationships. The objective is to estimate the function that best fits the training data while maintaining good generalization capabilities.

3.4.1.3 Random Forest

Random Forest is a technique that reduces the variance of prediction functions, known as bagging or bootstrap aggregation. It excels in handling high-variance, low-bias procedures, particularly decision trees. In regression tasks, multiple instances of the same regression tree are fitted to bootstrap-sampled versions of the training data, and their predictions are averaged.

Random Forest, introduced by Breiman in 2001, is a significant modification of bagging. It constructs a large collection of decorrelated trees and averages their predictions. On various problems, Random Forest demonstrates performance comparable to boosting while offering simpler training and tuning processes. (Hastie et al., 2009).

The essential idea in bootstrap aggregation is to average as many noisy but approximately unbiased models, and hence reduce the variance.

Mathematically, a Random Forest model can be represented as follows:

Given a training dataset with n observations, denoted as (x_1, y_1), (x_2, y_2),…, (x_n, y_n), where x_i represents the independent variables and y_i represents the corresponding dependent variable, the objective is to find the optimal function $f(x)$

Random Forest constructs B individual decision trees by applying the following steps:

For each tree $b = 1\ to\ B$:

1. Draw a bootstrap sample Z_b of size N from the training data.
2. Grow a decision tree T_b using the bootstrapped data, recursively splitting nodes until a stopping criterion is reached.

The Random Forest predictor for regression is defined as:

$$\hat{f}_{RF}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b\,(x)$$

In this equation, $\hat{f}_{RF}(x)$ represents the predicted value for a given input x. Each individual decision tree T_b in the Random Forest contributes its prediction $T_b(x)$. The predictions of all the trees are then averaged by taking the summation Σ across all trees and dividing by the total number of trees B. This averaging process helps to reduce the impact of individual noisy or biased predictions from each tree, leading to a more robust and accurate prediction for the Random Forest as a whole (Hastie et al., 2009).

By averaging the predictions of multiple trees, Random Forest reduces the variance and provides more robust predictions. Each decision tree contributes to the overall prediction based on its individual characteristics and the randomness introduced during the training process. This ensemble approach enhances the performance of Random Forest, making it a popular choice for various predictive modeling tasks.

3.4.1.4 Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost), is a machine learning model used for both regression and classification tasks. It is an ensemble model that consists of multiple decision trees. Each tree in the ensemble represents a weak learner that individually makes relatively simple predictions. However, due to the collective power of these weak learners, XGBoost is able to learn from a larger number of past errors compared to other models. By combining the predictions of these weak learners, the XGBoost model generates a final prediction for a given sample.
Mathematically, the XGBoost model can be represented as follows:

Given a training dataset with N samples and M features, denoted as $\{(x_i, y_i)\}$, where x_i represents the feature vector and y_i represents the corresponding target value, the XGBoost model is an ensemble of K decision trees, where each tree is denoted as $f_k(x)$ and represents a weak learner.

The prediction of the XGBoost model is given by:
$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$,

Where $\hat{y}_i$ represents the predicted value for the i-th sample.

To train the XGBoost model, an objective function is defined based on the training data and the current ensemble of trees. The objective function consists of two components: the loss function and the regularization term.

The objective function for the XGBoost model is defined as:
$Obj(\Theta) = \mathrm{L}(\mathrm{y}, \sum_{k=1}^{K} f_k(x)) + \Omega(\Theta)$,

where Θ represents the set of model parameters, $\mathrm{L}(\mathrm{y}, \sum_{k=1}^{K} f_k(x))$ is the loss function, and $\Omega(\Theta)$ is the regularization term.

During the training process, the XGBoost model iteratively adds new decision trees to the ensemble. Each tree is trained to minimize the gradient of the loss function with respect to the predicted values of the current ensemble.

The gradient boosting algorithm involves the following steps for each iteration:

1. Compute the negative gradient (residuals) of the loss function with respect to the current predictions.

2. Fit a new decision tree to the negative gradient using gradient histogram optimization. This technique approximates the optimal structure of the tree by finding the best split points for each feature using histograms.

3. Update the ensemble by adding the new tree, using a learning rate η to control the contribution of the new tree to the overall prediction.

4. Repeat steps 1-3 until the desired number of trees K is reached.

The XGBoost model incorporates additional techniques, such as column subsampling (feature subsampling), row subsampling (data subsampling), and regularization terms

(e.g., L1 and L2 regularization), to enhance its performance and prevent overfitting (Chen & Guestrin, 2016).

3.4.1.5 LSTM

Recurrent neural networks are well-suited for sequential and time-dependent data analysis. They excel at tasks where capturing dependencies across different time steps is crucial. RNNs, however, face a significant challenge known as the vanishing gradient problem. As information propagates backward through time during training, the gradients can become exponentially small, hindering the network's ability to learn long-term dependencies. This limitation restricts the RNN's capacity to model complex sequential patterns. To address this issue, a specialized type of RNN called Long Short-Term Memory (LSTM) can be used (Géron, 2019).

LSTM networks are a specialized form of architecture within RNNs that excel at capturing and preserving information over long sequences. Unlike basic RNNs, which rely on multiplying their memory by a weight matrix at each time step, LSTMs incorporate a dedicated long-term memory component called the memory cell (c). The memory cell is recurrently copied from one time step to another, allowing new information to be added incrementally without the risk of multiplicatively accumulating gradients over time.

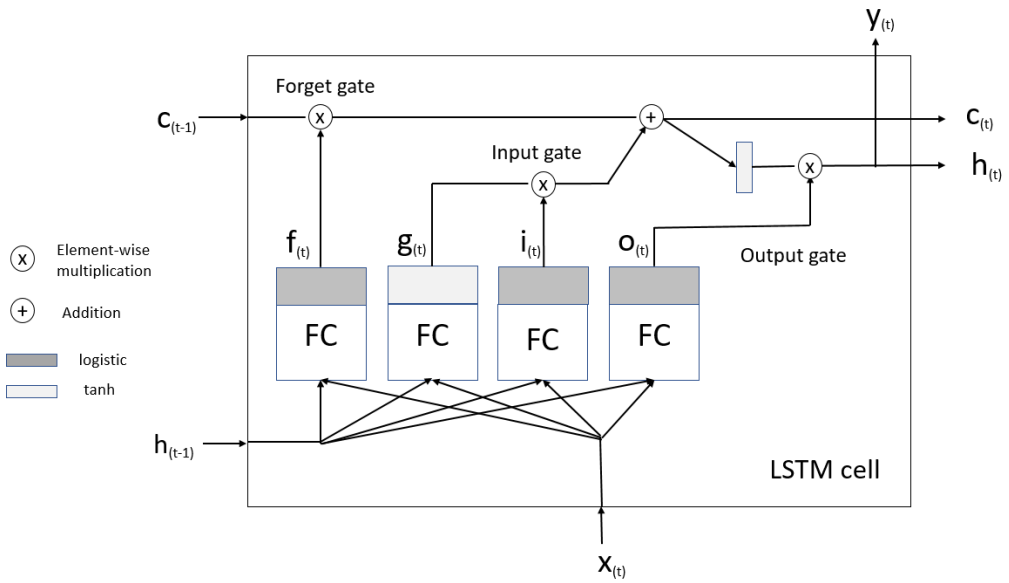

Figure 8 LSTM cell (adapted from Géron, 2019)

LSTMs leverage gating units to control the flow of information within the network. These gating units, represented as vectors, modulate the information passing through the LSTM via elementwise multiplication with corresponding information vectors. Mathematically, a LSTM model can be represented as follows: Given an input sequence of length T, an LSTM network consists of a series of memory cells. Each memory cell has its own set of parameters, including weight matrices (W) and bias vectors (b). The equations for a single memory cell are as follows:

1. Forget Gate:

$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$,

Here, f_t represents the forget gate activation at time step t. It determines whether each element of the memory cell should be retained (remembered) or reset to zero (forgotten) at the current time step. σ represents the activation function used for the forget gate, W_f represents the weight matrix associated with the previous hidden state h_{t-1} and the current input x_t for the forget gate, and b_f represents the bias for the forget gate.

2. Input gate

$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$,

$\hat{C}_t = g(W_C * [h_{t-1}, x_t] + b_C)$,

Where i_t represents the input gate activation at time step t. It regulates the extent to which memory cell element should be updated by incorporating new information from the input vector at the current time step. $\hat{C}_t$ represents the candidate cell state at time step t, which is a candidate update to the cell state. g represents the activation function used for the candidate cell state. W_i and W_C represents the weight matrices associated with the previous hidden state h_{t-1} and the current input x_t for the input gate and the candidate cell state, respectively. b_i and b_C represent the bias vectors for the input gate and the candidate cell state, respectively.

3. Update cell state:

$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t,$

Where C_t represents the cell state at time step t, which is updated based on the forget output f_t and the input gate output i_t. $\odot$ represents elementwise multiplications. C_{t-1} represents the previous cell state.

4. Output gate:

$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o),$
$h_t = o_t \odot h(C_t),$

Where o_t represents the output gate activation at time step t. It governs whether each memory cell element should be transferred to the short-term memory, which plays a role similar to the hidden state in basic RNN's. h_t represents the hidden state at time step t. h represents the activation function used for the hidden state. W_o represents the weight matrix associated with the previous hidden state h_{t-1} and the current input x_t for the output gate. b_o represents the bias vector for the output gate.

In these equations, the subscript t represents the time step, h_t represents the hidden state of the LSTM cell at time step t, and x_t represents the input at time step t. The weight matrices W and bias vectors b capture the network's learnable parameters (Géron, 2019).

There are four common activation functions used in LSTM models that can be used as input in the equations above, these are mainly; sigmoid activation function, hyperbolig tangent activation function, rectified linear unit, and softmax activation function which are explained in Chapter 2.2.4.2 Activation function.

3.4.2 Model optimization

Model optimization plays a crucial role in improving the performance and effectiveness of learning algorithms. When training a model, we often encounter hyperparameters that need to be set to appropriate values to achieve optimal results. Hyperparameters are external factors that govern the behavior and performance of a learning algorithm, such as the regularization parameter, learning rate, or kernel type.

I will discuss the process of model optimization and specifically focus on hyperparameter tuning. Hyperparameter tuning involves finding the best combination of hyperparameter values for a given learning algorithm to achieve optimal performance on a specific task or dataset. By carefully selecting suitable hyperparameter values, we can enhance the predictive accuracy, robustness, and generalizability of the models.

Linear Regression

For the linear regression models, I have chosen four hyperparameters that will be used to tune the models' performance. These hyperparameters are:

1. ***fit_intercept***:
    - Specifies whether to calculate the intercept for the linear regression model.
    - When set to False, the model assumes that the data is already centered and does not include an intercept term in calculations.
    - Adjusting this hyperparameter can be useful when dealing with data that is already centered or when you want to explicitly exclude the intercept from the model.

2. ***copy_X:***
    - Determines whether to create a copy of the input data (`X`).
    - If set to True, a copy of `X` is made, ensuring that the original data is not modified during the fitting process.

- Setting this hyperparameter to False can be useful when memory usage is a concern or when you are certain that the input data can be safely overwritten.

3. ***n_jobs:***
    - Controls the number of parallel jobs to use for computation.
    - This hyperparameter provides speedup when dealing with large problems, particularly when `n_targets > 1` (multiple target variables) or when `X` is sparse.
    - The default value of `None` means using a single job, unless in a `joblib.parallel_backend` context.
    - Setting `n_jobs` to -1 utilizes all available processors for parallel execution. Refer to the Glossary for more details.

4. ***positive:***
    - Enforces positive coefficients in the linear regression model.
    - If set to True, the model constrains the coefficients to be positive, applicable only for dense arrays (non-sparse data).
    - This hyperparameter is useful when you want to impose positive constraints on the coefficients, which can be relevant in certain applications or domains
      (scikit-learn, n.d. a).

Support Vector Regression

For Support Vector regression models, I have chosen ten hyperparameters that will be used to tune the models' performance:

1. ***C***: The regularization parameter, C, controls the trade-off between achieving a smaller training error and allowing more deviations in the solution. It influences the width of the margin and the number of support vectors used for fitting the model. Different values of C can lead to varying degrees of model complexity and generalization capability.

2. ***coef0***: The coef0 parameter determines the influence of the non-linear terms in the model. It affects the shape of the decision function and can impact the model's ability to capture complex relationships between the features and the target variable.

3. ***degree***: The degree parameter is specific to polynomial kernels and defines the degree of the polynomial function used to transform the input data. It controls the complexity and flexibility of the model's decision function.

4. ***epsilon***: Epsilon, also known as the epsilon-insensitive loss parameter, sets the margin around the regression line within which no penalty is associated with errors. It determines the tolerance for errors in the training data and affects the model's sensitivity to deviations from the target variable.

5. ***gamma***: Gamma defines the kernel coefficient for 'rbf', 'poly', and 'sigmoid' kernels. It influences the shape and reach of the decision boundary and controls the influence of individual training samples on the model. Different gamma values can result in different levels of model flexibility and overfitting.

6. ***kernel***: The kernel parameter specifies the type of kernel function used for transforming the input data into a higher-dimensional feature space. SVR supports various kernel functions such as linear, polynomial, radial basis function (RBF), and sigmoid. Each kernel has different characteristics and can capture different types of relationships between the features and the target variable.

7. ***max_iter***: The maximum number of iterations allowed for convergence. It determines the maximum number of iterations the model will perform during training. Setting an appropriate value is important to ensure convergence without excessive computation.

8. ***shrinking***: The shrinking parameter determines whether to use the shrinking heuristic. When set to True, the model applies a shrinking strategy to speed up the training process by eliminating some support vectors. However, this may slightly affect the model's performance.

9. ***tol***: The tolerance for stopping criterion. It specifies the desired precision of the solution. A lower tolerance value can lead to more accurate solutions but may increase computation time.

10. ***verbose***: The verbosity parameter controls the amount of output information displayed during the model's training process. Setting it to False suppresses the output, while setting it to True provides detailed information about the training progress.
    (scikit-learn, n.d. b).

Extreme Gradient Boosting (XGBoost)

I have chosen the following six hyperparameters to tune the Extreme Gradient Boosting models' performance:

1. ***max_depth***: This hyperparameter controls the maximum depth of each tree in the boosting process. Increasing `max_depth` allows the model to capture more complex relationships in the data but can also lead to overfitting. On the other hand, reducing `max_depth` can simplify the model and enhance its generalization ability.

2. ***n_estimators***: The number of boosting iterations or decision trees in the model is determined by the `n_estimators` hyperparameter. Increasing this value can improve the model's performance until a certain point of diminishing returns is reached. Adding more trees may result in longer training times, so it is important to strike a balance between model performance and computational efficiency.

3. ***learning_rate***: The learning rate determines the step size at each boosting iteration. A smaller learning rate makes the model converge more gradually, allowing for finer adjustments and potentially better generalization. However, using a very small learning rate may require more boosting iterations to achieve optimal performance.

4. ***subsample***: This hyperparameter controls the fraction of samples used for training each tree. Setting `subsample` to a value less than 1.0 introduces stochasticity into the model and can help prevent overfitting. It is common to set `subsample` to a value below 1.0, such as 0.5 or 0.7, to improve the model's robustness.

5. ***colsample_bytree***: This hyperparameter determines the fraction of features (columns) to be randomly sampled for each tree. By selecting a subset of features, the model can focus on the most informative ones, reducing the risk of overfitting and improving generalization. Values between 0.5 and 1.0 are commonly used for this hyperparameter.

6. ***min_child_weight***: This hyperparameter sets the minimum sum of instance weights (hessian) required to further partition a leaf node in the tree. Increasing `min_child_weight` can help control overfitting by adding regularization. A larger value results in a more conservative model. (Chen & Guestrin, 2016).

Random Forest

Five hyperparameters are chosen to tune the Random Forest models' performance:

1. ***max_depth***: The max_depth hyperparameter controls the maximum depth of the decision trees in the RF ensemble. Increasing the max_depth allows the trees to capture more complex relationships in the data. However, a higher

max_depth may lead to overfitting, so it is crucial to find the right balance to ensure optimal model performance.

2. ***max_features***: This hyperparameter determines the number of features to consider when looking for the best split at each tree node. The auto option considers all features, while sqrt uses the square root of the total number of features. Choosing the appropriate value for max_features can prevent overfitting and increase the model's ability to generalize to unseen data.

3. ***min_samples_leaf***: The min_samples_leaf hyperparameter specifies the minimum number of samples required to be at a leaf node. A smaller value allows the trees to capture more specific patterns in the data, potentially leading to overfitting. On the other hand, a larger value promotes a more generalized model. It is crucial to find the right balance to avoid underfitting or overfitting.

4. ***min_samples_split***: This hyperparameter sets the minimum number of samples required to split an internal node. Similar to min_samples_leaf, a smaller value leads to more specific splits, while a larger value promotes more generalized splits. Careful tuning of min_samples_split is necessary to strike the right balance between model complexity and generalization.

5. ***n_estimators***: The n_estimators hyperparameter defines the number of trees in the RF ensemble. Increasing the number of estimators can improve the model's performance until reaching a point of diminishing returns. However, a higher number of estimators also increases the computational cost, so it is essential to consider the trade-off between performance and efficiency. (scikit-learn, n.d. c).

LSTM

I have chosen seven hyperparameters for tuning the LSTM model's performance:

1. ***units***: The units hyperparameter determines the dimensionality of the output space of the LSTM layer. Increasing the number of units allows the model to capture more complex patterns in the data but also increases the model's computational complexity. It is important to find the right balance between model complexity and efficiency.

2. ***dropout_rate***: Dropout is a regularization technique that helps prevent overfitting in neural networks. The dropout_rate hyperparameter determines the fraction of the input units to drop during training. Using dropout introduces randomness into the model and can improve its generalization ability. Tuning the dropout_rate can help find the optimal level of regularization for the LSTM model.

3. ***num_layers***: The num_layers hyperparameter defines the number of LSTM layers in the model. Adding more layers increases the model's capacity to capture complex dependencies but also makes it more prone to overfitting. Finding the appropriate number of layers is crucial to balance model complexity and generalization.

4. ***epochs***: The epochs hyperparameter specifies the number of times the entire training dataset is passed through the LSTM model during training. Increasing the number of epochs allows the model to learn more from the data but can lead to overfitting if not controlled. It is essential to monitor the model's performance on a validation dataset and choose an appropriate number of epochs.

5. ***batch_size***: The batch_size hyperparameter determines the number of samples to be propagated through the LSTM model at once. It affects the model's training speed and memory usage. Smaller batch sizes introduce more noise

into the training process but can help the model converge faster. Larger batch sizes provide a smoother gradient estimate but require more memory. The batch_size should be chosen based on the available computational resources and the characteristics of the dataset.

6. ***optimizer***: The optimizer hyperparameter determines the algorithm used to update the weights and biases of the LSTM model during training. It plays a crucial role in the model's learning process and affects the speed and effectiveness of convergence. Different optimizers, such as Stochastic Gradient Descent (SGD), Adam, RMSprop, and Adagrad, offer varying characteristics and performance. Experimenting with different optimizers can help identify the one that works best for the specific problem and dataset, striking a balance between convergence speed and accuracy.

7. ***activation***: The activation hyperparameter specifies the activation function to be used in the LSTM layers. Common choices include 'relu', 'sigmoid', and 'tanh'. The choice of activation function can impact the model's ability to capture nonlinear patterns in the data. Experimenting with different activation functions can help find the one that works best for the specific problem. (Keras, n.d.)

By systematically exploring different combinations of hyperparameters for each model, we can evaluate the performance of the models using appropriate evaluation metrics and select the optimal set of hyperparameters that yield the best performance on the validation dataset.

To optimize these hyperparameters, a grid search strategy was employed. This involved defining a grid of potential parameter values and exhaustively evaluating the model's performance for each combination. The grid search procedure, coupled with cross-validation techniques, allowed for robust evaluation and comparison of the models' performance across different stocks. Through the hyperparameter optimization process, we aimed to strike a balance between model complexity and

generalization. A too complex model can lead to model overfit, which occurs when a model tries to explain small variations in the data. A model that overfits, predicts good on training data, but performs poorly on new unseen patterns (Burkov, 2019).

3.4.3 Evaluating model performance

When evaluating the numeric outcomes of prediction models, it is important to evaluate the effectiveness of the models using a measure of accuracy. There are several measures that can be used to evaluate the predictions, each method with its own nuance. Some of the typical measures used in machine learning evaluations are the proportion of correct predictions, F1-score, sensitivity and many more. However, these metrics are used for classification problems. When measuring regression problems, you have to use quantitative measures of performance and the most common methods are the root mean squared error (RMSE), the mean squared error (MSE), the coefficient of determination or "R-squared" (R^2) (Kuhn & Johnson, 2013). The mean absolute error (MAE), and the mean absolute percentage error (MAPE) (Hyndman & Athanasopoulos, 2018).

Coefficient of determination ($\boldsymbol{R^2}$)

The coefficient of determination is a statistical measure that represents the proportion of variance in the dependent variable that can be explained by the independent variables in a regression model. It indicates how well the regression model fits the observed data. The value of R^2 ranges from 0 to 1, where a higher value indicates a better fit.

The mathematical representation of R^2 is:

$$R^2 = 1 - \frac{SSR}{SST},$$

Where SSR is the sum of squared residuals, and SST is the total sum of squares.

R^2 measures the proportion of the variability in the dependent variable that is explained by the regression model. A value of 0 indicates that the model does not explain anything of the variability, while a value of 1 explains all the variability.

It is worth mentioning that R^2 has some limitations and should be interpreted with caution. It does not determine whether the regression model is good or bad, but rather it measures the goodness of fit, and it should rather be used alongside with other evaluation metrics to understand the results better (Kuhn & Johnson, 2013).

Mean Squared Error (MSE)

MSE is an evaluation metric commonly used in regression models to measure the average squared difference between the predicted values and the actual values of the target variable. It provides a quantitative measure of the average magnitude of the residual error. A lower MSE value indicates that the model has smaller average squared differences between the predicted and actual values, which suggests better overall performance.

The mathematical representation of MSE is:

$$MSE = \frac{1}{n} * \sum(y_i - \hat{y}_i)^2,$$

Where n is the number of samples or observations, y_i represents the actual values of the target variable, and $\hat{y}_i$ is the predicted values of the target variable.

MSE provides a measure of the average magnitude of the squared errors between the predicted and actual values. It gives more weight to larger errors due to the squaring operation, making it particularly sensitive to outliers. MSE itself is not in the original unit of the target variable, which can make it less interpretable compared to other metrics like RMSE (Root Mean Squared Error) (Kuhn & Johnson, 2013).

Root mean squared error (RMSE)

RMSE is an evaluation metric used for regression models that quantifies the average magnitude of the residual errors. RMSE is expressed in the same units as the target variable, which makes it interpretable and comparable across different datasets. RMSE is calculated by taking the square root of the mean of the squared differences between the predicted values and the actual values.

The formula for calculating RMSE is:

$$RMSE = \sqrt{MSE},$$

RMSE is beneficial because it penalizes larger errors more heavily due to the squaring of differences. It provides a measure of how well the model's predictions fit the observed data, where lower values indicate a better performance (Kuhn & Johnson, 2013).

Mean Absolute Error (MAE)

MAE is typically used in regression problems to measure the average absolute difference between the predicted values and the actual values. MAE is expressed in the same units as the target variable, which makes it easy to interpret. A lower MAE indicates better performance, as it shows that the model's predictions are closer to the actual values on average.

The mathematical representation of MAE is:

$$MAE = \frac{1}{n} * \sum_{i=1}^{n} |y_i - \hat{y}_i|,$$

Where n is the number of samples, y_i is the actual values of the target variable, $\hat{y}_i$ is the predicted value of the target variable, $|y_i - \hat{y}_i|$ represents the absolute of the difference between the actual value and the predicted value across all datapoints. The absolute difference is taken for each data point and then averaged across all the data points to obtain the mean absolute error.

Unlike other error metrics that squares the error, such as RMSE, MAE considers the absolute difference between the predicted and actual values, making it less sensitive to outliers (Hyndman & Athanasopoulos, 2018).

Mean Absolute Percentage Error (MAPE)

MAPE is used to evaluate the accuracy of a forecasting model, typically in the context of time series analysis. MAPE measures the average percentage difference between the predicted and actual values, relative to the actual values.

The mathematical representation of MAPE is:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| * 100,$$

MAPE is expressed as a percentage, indicating the average percentage deviation between the predicted and actual values. It provides a relative measure of the forecasting accuracy, allowing comparisons across different datasets or models. MAPE is useful in situations where the magnitude of the error is important and needs to be considered in a percentage form. However, it has some limitations. MAPE becomes undefined or infinite when the actual values are zero or close to zero, and it tends to emphasize larger errors due to the percentage calculation. Therefore, it is important to interpret MAPE in conjunction with other evaluation metrics and consider the specific characteristics of the dataset and the specific context of the problem (Hyndman & Athanasopoulos, 2018).

4.0 Results

I will in this chapter present the results and key findings of the analyses. First descriptive statistics from each model will be presented, and thereafter aggregated results will be examined to investigate the overall performance of each model across the stocks. Important findings from each model will be highlighted, followed by a comparative analysis of the aggregated results from the models.

4.1 Descriptive statistics

Table 1 contains the results from the evaluation metrics for all models across all stocks. I will start by presenting the results of each individual model and highlight interesting findings.

Table 1 Evaluation metric scores

Individual Evaluation Metrics for all models and all stocks

Stock	Evaluation Metrics	OLS	SVR	RF	XGB	LSTM
Equinor	R_squared	-0,198	-2,340	0,170	0,120	0,520
	MSE	7144,465	662,470	346,750	363,760	133,240
	RMSE	84,525	21,390	13,140	13,480	10,040
	MAE	63,245	16,910	9,360	9,760	9,040
	MAPE	27,73 %	16,80 %	10,82 %	11,46 %	9,24 %
Mowi	R_squared	-4,839	-106,71	0,490	-0,040	0,170
	MSE	3699,583	9633,430	109,630	221,780	198,470
	RMSE	60,824	82,620	7,680	12,070	10,270
	MAE	56,762	82,030	6,600	10,600	9,600
	MAPE	29,98 %	435,36 %	22,83 %	26,41 %	18,32 %
Telenor	R_squared	-11,141	-0,480	-0,150	-0,170	0,650
	MSE	1513,926	123,820	162,660	166,370	22,750
	RMSE	38,909	9,580	9,790	10,180	4,480
	MAE	30,105	7,890	7,490	7,780	3,890
	MAPE	26,59 %	19,30 %	12,35 %	13,03 %	7,41 %
DNB	R_squared	0,994	-2,660	-0,350	-0,510	0,530
	MSE	5,246	324,140	136,310	151,320	26,880
	RMSE	2,290	13,800	9,400	10,110	4,360
	MAE	1,593	12,050	7,410	8,020	3,960
	MAPE	1,22 %	27,23 %	14,36 %	15,94 %	8,77 %

Aggregated Evaluation Metrics across all stocks

Evaluation Metrics	OLS	SVR	RF	XGB	LSTM
R_squared	-3,796	-28,048	0,040	-0,150	0,468
MSE	3090,805	2685,965	188,838	225,808	95,335
RMSE	46,637	31,848	10,003	11,460	7,288
MAE	37,926	29,720	7,715	9,040	6,623
MAPE	21,38 %	124,67 %	15,09 %	16,71 %	10,94 %

4.2 Linear regression model

Individual stock analysis

The linear regression model is a benchmark model and is not expected to perform very well. This is initially true, except for the case of the stock "*DNB*", which I will discuss in the findings.

The R^2 results from the stocks; *"Equinor, Mowi, and Telenor"* are respectively "*-0.198, -4.839, and -11.141*", while the model provides a better R^2 score for Equinor and Mowi, than for Telenor. The R^2 scores are still negative, indicating that the OLS models do not capture the variability of the data very well. In other words, it suggests that there may be additional factors influencing the dependent variable besides the independent variable, which could be due to lacking relevant features or underfitting. This is expected as other factors can influence the development of stock prices besides historical price data. However, there might be more patterns in the historical price, which the OLS model is not able to capture since it does not follow a linear pattern.

Moreover the MSE results are respectively "*7144.465, 3699.583, and 1513.926*", indicating that there is a substantial difference in the average squared differences between the predicted and actual values. On average, the model's predicted values deviate significantly from the true values. By analyzing the $RMSE$, we can see the results in a more explicit context. Specifically, the $RMSE$ scores are "*84.525, 60.824, and 38.909*" which indicate that the model is on average off by *84,525 NOK* for Equinor, *60,824 NOK* for Mowi, and *38,909 NOK* for Telenor based on the average squared differences. The magnitude of the error of these results depends on the range of the target variable. However, since the average adjusted closing price for the stocks in the prediction window were respectively "*208.77 NOK, 189.01 NOK, and 119.48 NOK*", it would be safe to assume that this average error is too far off to be considered an accurate model, even though outliers may affect these estimates to a certain degree.

Like the RMSE results, the MAE results can be compared directly to the inputs and yield a similar outcome. The key difference is that MAE represents the average absolute error instead of the average squared differences. In simple terms, they both measure prediction errors, but RMSE is more sensitive to outliers compared to MAE. The MAE results are respectively "*63.245, 56.762, and 30.105*", again indicating that the predictions are far off compared to the actual values. Lastly, the $MAPE$ values, which represent the average absolute percentage error in the model's predictions, indicate that the predictions are off by respectively "*27.73%, 29.98%, and 26.59%*".

Aggregated stock analysis

On average across all the stocks the OLS models' evaluation metrics "$R^2, MSE, RMSE, MAE, and\ MAPE$" have the following results: *"-3.796, 3090.805, 46.637, 37.926, and 21.38%*", respectively. The results indicate that on average the model has a negative fit, and on average it is off by roughly 21%. The results indicate a poor model fit, which is to be expected from a linear regression model in predicting stock prices.

Key findings

An interesting finding in the OLS model is the predictions of the DNB stock. Based on the evaluation metrics "$R^2, MSE, RMSE, MAE, and\ MAPE$", the results are "*0.994, 5.246, 2.290, 1.593, and 1.22%*", respectively. The R^2 results indicate a near perfect fit for the model, and the prediction estimates are only off by 1.22%, which seems remarkable. Figure 9 provides a visual presentation of the results.

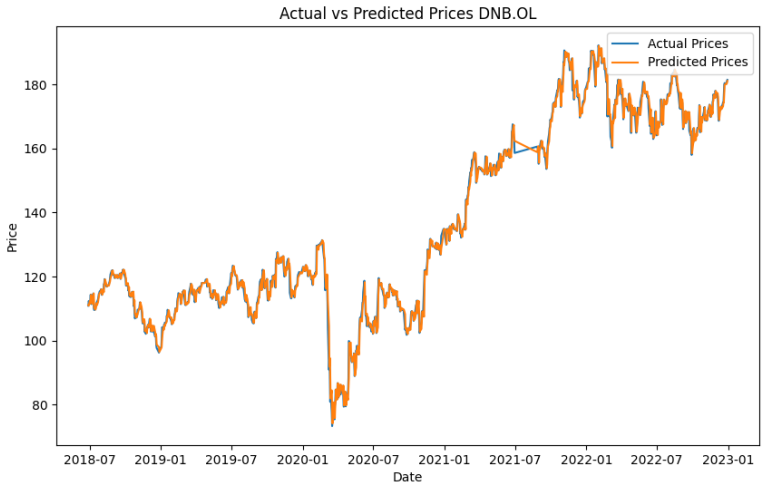

Figure 9 OLS predictions compared to actual prices of DNB

However, it is important to consider these results with caution. It is uncommon for any model to accurately predict the complexities associated with stock price development with such precision.

4.3 Support vector regression model

While OLS linear regression models may struggle to effectively predict non-linear data, SVR models are expected to excel in such scenarios. In the context of stock prices, which often exhibit complex patterns and outliers, OLS models are typically sensitive to outliers. On the other hand, SVR's ability to model non-linear relationships and its robustness against outliers enhance its predictive capabilities compared to OLS linear regression.

Individual stock analysis

The R^2 results for the stocks "*Equinor, Mowi, Telenor, and DNB*" are respectively "*-2.340, -106.71, -0.480, and -2.660*". Surprisingly, the SVR models yield worse R^2 scores on all stocks except for Telenor, indicating that the SVR model is less effective at capturing the variability of the data compared to the OLS model. A potential explanation for these results is that the stock data exhibits a more linear relationship than expected. However, these results do not necessarily mean that the

model is worse at predicting stocks, but rather that there may be factors affecting stock price development that are not included in the features. Nonetheless, the R^2 score alone does not provide a perfect evaluation of the predictions, so let us examine the other evaluation metrics.

The MSE results are respectively "*662.470, 9633.430, 123.820, and 324.140*". Compared to the OLS model, SVR performs better for Equinor and Telenor, but worse for DNB. These results still indicate that, on average, the model's predicted values deviate significantly from the true values. By analyzing the RMSE scores, we can interpret the mean squared error in a format more closely related to the actual values. The RMSE scores are "*21.390, 82.620, 9.580, and 13.800*", indicating that, on average, the model's predictions are off by *21.390 NOK* for Equinor, *82.620 NOK* for Mowi, *9.580 NOK* for Telenor, and *13.800 NOK* for DNB based on the average squared differences. It is important to note that the magnitude of these errors depends on the range of the target variable, and for Mowi, the prices are still quite far off. However, for Equinor, Telenor, and DNB, the prices are closer, suggesting an improvement in model performance.

The MAE results provide similar observations, and while the results for Mowi are still off by "*82.030 NOK*", the errors for Equinor, Telenor, and DNB are decreasing with values of "*16.910 NOK, 7.890 NOK, and 12.050 NOK*" respectively. These results further suggest an improvement in model performance compared to the OLS model.

Finally, the MAPE results (in percentage) show errors of "*16.80%, 435.36%, 19.30%, and 27.23%*". These figures demonstrate that our model performs most accurately for Equinor, with an average deviation of only *16.8%* from the actual prices. However, the results for Mowi deviate significantly more compared to the OLS model.

Aggregated stock analysis

On average across all stocks, the evaluation metrics "R^2*, MSE, RMSE, MAE, and MAPE*" for the SVR model have the following results: "*-28.048, 2685.965, 31.848,*

29.720, and 124.67%". MSE, RMSE, and MAE outperform the OLS model, while R^2 and MAPE perform worse.

Key findings

The results from the evaluation metrics suggests that the model performs very poorly on the Mowi stock. However, by looking at the plot from the predicted values against the actual values of Mowi, we can identify an interesting trend.

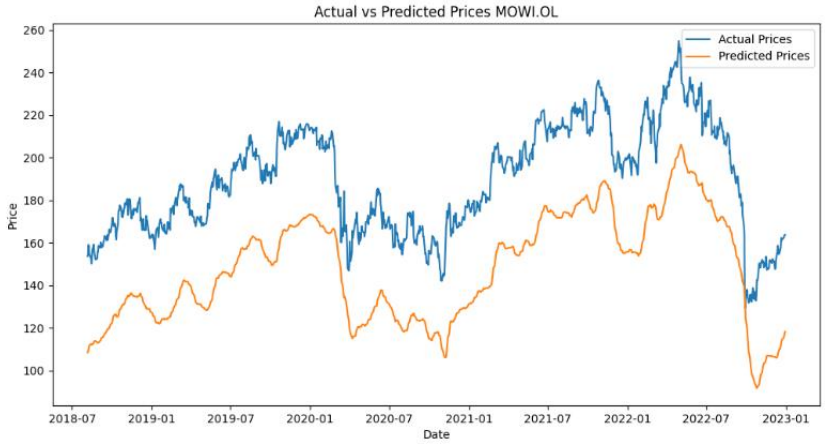

Figure 10 SVR predictions compared to actual prices of Mowi

The accuracy of the prediction is quite off. However, it seems as if the SVR model is still able to capture the trends in the data, even if it is not predicting the scale of the prices accurately.

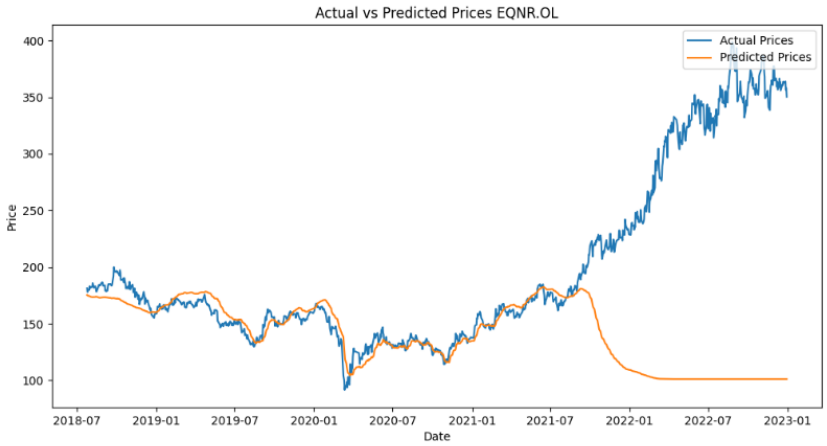

Figure 11 SVR predictions compared to actual prices of Equinor

Additionally, an interesting pattern occurs for the prediction of Equinor. From figure 11 we can see that the model identifies patterns up to a certain point in which it follows the opposite pattern of the actual prices until it slowly progresses into static values.

4.4 Random forest model

The SVR model is specifically suitable when the relationship between the input features and the target variable is expected to be non-linear and is robust against outliers, and stock prices are typically characterized by both non-linearity and outliers. The RF model is typically preferred when dealing with complex interactions between features. However, since this analysis only includes one feature, one would anticipate that the SVR model would perform better, but the RF model might be able to identify certain complex patterns in the data that the SVR model cannot identify.

Individual stock analysis

The R^2 results for the stocks "*Equinor, Mowi, Telenor, and DNB*" are respectively "*0.17, 0.49, -0.15, and -0.35*". Compared to the SVR model, it would seem like the RF model produces a higher R^2 for all the stocks. Indicating that the RF model is more effective at capturing the variability of the data compared to the SVR model.

The MSE results are respectively "*346.750, 109.630, 162.660, and 136.310*". Compared to the SVR model, the RF model performs better on all stocks except for the Telenor stock. This can indicate that on average, the model's predicted values deviate less than the SVR model. By analyzing the RMSE values, we get a clearer picture of what these deviations actually mean. The RMSE results are respectively "*13.14, 7.68, 9.79, and 9.4*". These results indicate that on average the model's predictions are off by *13.14 NOK* for Equinor, *7.68 NOK* for Mowi, *9.79 NOK* for Telenor, and *9.4 NOK* for DNB. The RF predictions have fewer errors in all the stocks except for Telenor, which indicate a progress in model performance.

Aggregated stock analysis

On average across all stocks, the evaluation metrics "R^2, *MSE, RMSE, MAE, and MAPE*" for the RF model have the following results: "*0.04, 188.838, 10.003, 7.715, and 15.09%"*. All metrics outperform the SVR model.

Key Findings

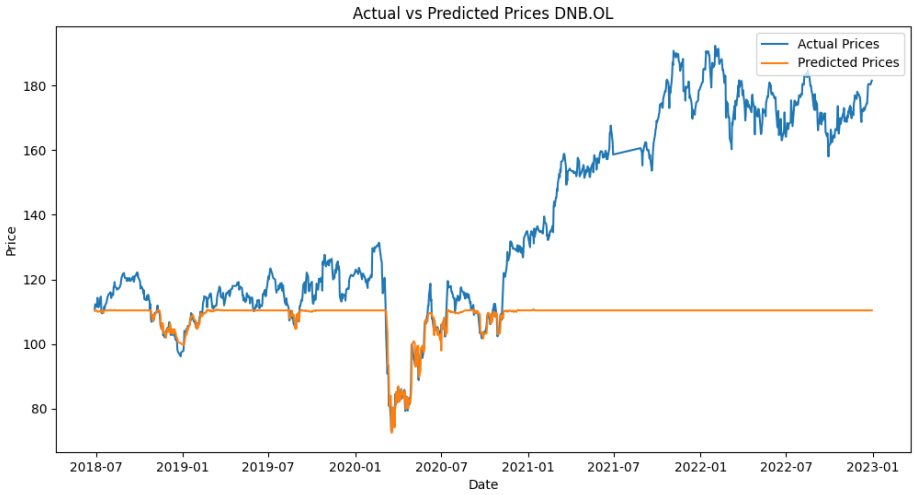

Figure 12 RF predictions compared to actual prices of DNB

An interesting finding from the predictions of the RF model, is the predictions of the DNB stock. From the predicted values, it would seem like the predictions have a kind of upper threshold. When the actual values exceed roughly 110 NOK, the predictions becomes static. A possible explanation for this finding could be the distribution of the adjusted closing price might change as it reaches or exceeds 110 NOK. In other words, the underlying patterns might exhibit a shift in linearity after a certain threshold and the model is not able to identify this pattern.

4.5 XGBoost

Random Forest and XGBoost are both ensemble learning methods that combine multiple decision trees. The main difference is how they build the ensemble. In Random Forest, the trees are trained independently, and their predictions are averaged or voted to make the final prediction. XGBoost, on the other hand, builds the trees sequentially, with each tree trying to correct the mistakes of the previous trees using gradient descent optimization.

Individual stock snalysis

The R^2 results for the stocks "*Equinor, Mowi, Telenor, and DNB*" are respectively "*0.120, -0.040, -0.170, and -0.510*". Compared to the RF model, the XGB model produced a lower R^2 score for all stocks. This indicates that that the XGB model is less effective in capturing the variability of the data compared to the RF model. These results could be due to the nature of the data. As stated, the RF model is more robust to noisy data and outliers. And since stock data can potentially be very noisy, the RF model could handle these patterns better than the XGBoost model.

The MSE results are respectively "*363.760, 221.780, 166.370, and 151.320*". Compared to the RF model, the XGBoost model has on average a slightly worse squared error, which could indicate that the model's predicted values on average deviate more than the values of the RF model. By analzying the RMSE results: "*13.480, 12.070, 10.180, and 10.110*", we can see that the same patterns exist here. The RMSE results indicate that the XGBoost predictions are off by *13.48 NOK* for Equinor, *12.07 NOK* for Mowi, *10.18 NOK* for Telenor, and *10.11* NOK for DNB. These results also indicate that on average, the XGB model have more errors in the stock prediction on all stocks compared to the RF model.

If we analyze the MAE values, which is less sensitive to ouliers than RMSE values, we still see the same indications, with the following values: "*9.760, 10.600, 7.780, and 8.020*". The values are still better than the OLS and SVR model, but it seems as the XGB model performs worse than the RF model.

Aggregated stock analysis

On average across all stocks, the evaluation metrics "R^2, *MSE, RMSE, MAE, and MAPE*" for the XGB model have the following results: *"-0.150, 225.808, 11.460, 9.040, and 16.71 %".* All metrics perform worse than the RF model.

Key findings

Although the RF model provided better results, it would seem like both models follow roughly the same pattern, the only difference is that the RF model provides slightly better results.

4.6 LSTM

LSTM models are specifically designed for handling sequential and time-series data. The LSTM model is well-suited for capturing patterns and dependencies in temporal sequences, making it specifically good at handling time series forecasting, and the model is therefore expected to perform well on stock prediction compared to the other models.

Individual stock analysis

The R^2 results for the stocks "*Equinor, Mowi, Telenor, and DNB*" are respectively "*0.52, 0.17, 0.65, and 0.53*". Compared to the XGB model, the LSTM model produced a higher R^2 score for all the stocks. In fact, it produces a higher R^2 score across all models, indicating that the LSTM model is especially effective at capturing the variability of the data.

The MSE results are respectively "*133.24, 198.47, 22.75, and 26.88*" which are the best results across all models, having the lowest squared error across the models. The LSTM model's predictions deviate less from the actual prices than the other models. By analyzing the RMSE results "*10.04, 10.27, 4.48, and 4.36*", we see that the same patterns emerge here, and the predicted prices deviate less compared to the other models. The RMSE results indicate that the LSTM model's predictions are off by *10.04 NOK* for Equinor, *10.27 NOK* for Mowi, *4.48 NOK* for Telenor, and *4.36 NOK* for DNB. For Equinor, Telenor, and DNB, these are the best RMSE results so far. However, the Random Forest model still produces better RMSE results for Mowi. By analyzing the MAE score, which is less sensitive to outliers than RMSE, we still see the same indications with the following values: "*9.04, 9.6, 3.89, and 3.96*" indicating a very low absolute error on average, but the Random Forest model produces better

MAE results for Mowi as well. Interestingly, the MAPE values do not show the same pattern, as showed by the following values: *"9.24 %, 18.32 %, 7.41 %, 8.77 %"*. The LSTM model produced the best MAPE results across models for all stocks.

Aggregated stock analysis

On average across all stocks, the evaluation metrics "R^2, *MSE, RMSE, MAE, and MAPE*" for the LSTM model have the following results: "*0.47, 95.34, 7.29, 6.62, and 10.94%*". In this aggregation, the LSTM model produces the overall best results across all metrics.

Key findings

A notable observation emerge from the model's predictions for Mowi and Telenor. Upon examining the plots, we notice that the model exhibits some ability to recognize the patterns present in the actual prices during the test period. However, it becomes apparent that the model suffers from a certain "lag," predicting the price developments after they have already occurred in some instances.

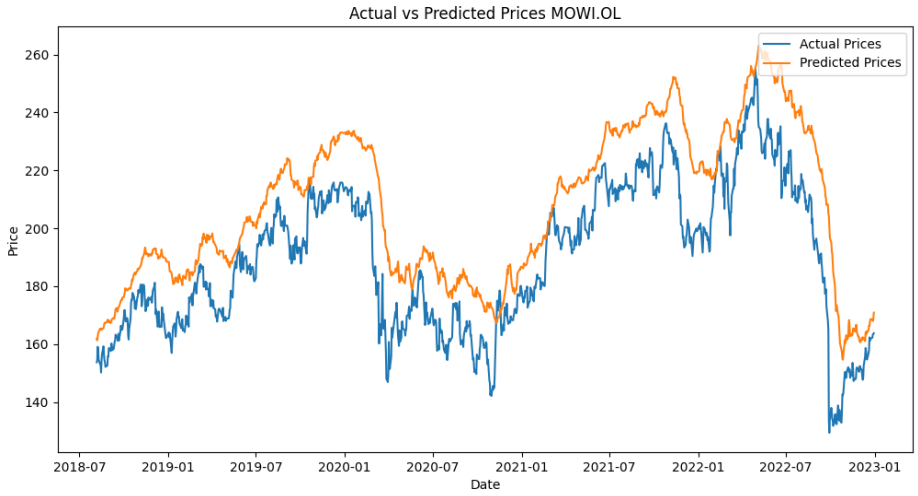

Figure 13 LSTM predictions compared to actual prices of Mowi

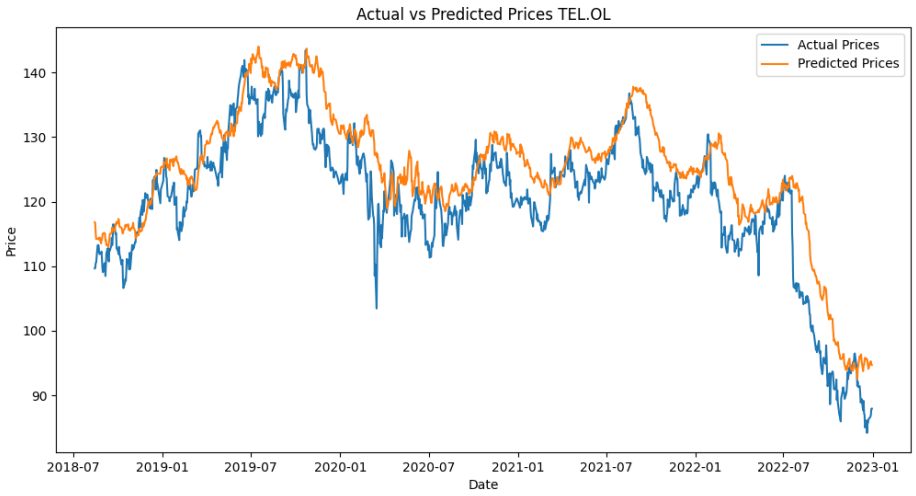

Figure 14 LSTM predictions compared to actual prices of Telenor

Based on these findings, as well as the scores from the aggregated evaluation metrics across all stocks in Table 1, it is evident that the LSTM model outperforms all other models across all metrics. Overall, the LSTM model is the preferred model to identify patterns in the Norwegian stock market.

4.7 Comparison of the different models

I will conduct a comparative analysis of the different models based on the results obtained from the individual analyses. I will examine the similarities, differences, and strengths exhibited by each model.

Similarities, differences, and strengths of the models

OLS (Ordinary Least Squares), SVR (Support Vector Regression), RF (Random Forest), XGB (XGBoost), and LSTM (Long Short-Term Memory) are all popular machine learning models used for various prediction tasks. While they have certain similarities, such as being supervised learning algorithms, they also have notable differences in their underlying principles and approaches.

The OLS model is specifically good at estimating the relationship between a dependent and independent variable when there is a linear relationship between the

two variables. This makes it a reasonable choice, when defining feature importance. However, since it assumes linearity in the relationship, it could be difficult to make accurate predictions when dealing with non-linear data, such as stock data. This seems to be the case from the results, where it is able to capture some patterns, but is not able to make accurate predictions. While it cannot necessarily be used to predict complex non-linear data, it can be used as a benchmark model, as well as establishing feature importance.

Just as the OLS, the SVR assumes a linear relationship between the input feature and the target variable. However, while the OLS model focuses on minimizing the sum of squared residuals, the SVR model aims to find a hyperplane that maximizes the margin while allowing for a certain tolerance of errors. This difference makes SVR better at capturing patterns of non-linearity, which is more ideal when dealing with time-series data.

While the SVR model is good at handling non-linearity, it has to be explicitly specified what type of kernel to use “e.g., linear, polynomial, radial basis function”. The random forest model, however, is able to handle non-linear data, without transforming the data. It achieves this by randomly selecting subsets of features and building trees based on these subsets, allowing for interactions between variables. Additionally, the SVR model seems to be more sensitive to outliers than the RF model based on the predictions. This would make the RF model better at predicting the adjusted closing prices when there is higher volatility which can occur with different frequencies on different stocks. The RF model is an ensemble machine learning model, which means that it uses predictions from several models and combine the predictions to improve the overall performance. This process makes the model computationally expensive.

Like Random Forest, XGBoost is also an ensemble machine learning model. However, they each have their individual strengths as well. From the predictions, both models follow similar prediction patterns, but the RF model has an overall higher prediction accuracy. Given the volatility of the data, this could indicate that the

RF model is better at handling outliers and noisy data. As with the Random Forest model, The XGBoost model is also computationally expensive.

The LSTM model consistently outperforms other models in predicting the adjusted closing price. It demonstrates robustness in handling outliers and effectively adapts to changing market trends. However, it requires careful hyperparameter tuning to avoid overfitting. This process can be computationally expensive. While the LSTM model delivers impressive results, there are instances where a simpler model may be preferred due to runtime considerations.

While all models share similarities as supervised learning algorithms, they exhibit distinct characteristics and strengths. OLS and SVR are effective for linear relationships, with SVR excelling in capturing nonlinear patterns in time-series data. RF handles nonlinear data without transformation and performs well in the presence of outliers, while XGB achieves high prediction accuracy through a gradient boosting framework. The LSTM model consistently outperforms others, demonstrating robustness in handling outliers and adapting to shifting market trends. However, it requires careful hyperparameter tuning and can be computationally expensive. Consideration should be given to the specific data characteristics and runtime requirements when selecting the appropriate model for stock price prediction.

5.0 Discussion and implications

In this part of the thesis, I will begin by discussing the results, where I have divided the discussion into three categories: stock patterns, overfit, and efficient market hypothesis. I will also go through any limitations of my approach, and lastly, I will discuss relevant future research.

5.1 Discussion of results

Stock patterns

For the SVR model, there are two interesting findings. One of the findings is for the stock “Mowi”. The results from the evaluation metrics suggests that the model performs poorly. However, when looking at the plot from the predicted values against the actual values, we see that the model is still able to capture the trends of the data, even if it is not predicting the scale of the pries accurately. A possible explanation for this behavior can be traced to the features of the data. Since the model is only trained on one feature “adjusted closing price”, it might indicate that the model is able to identify complex patterns in the historical price. However, there might be additional external factors affecting the development that the SVR model is not able to identify, and therefore, the scale of the data is wrong.

Additionally, an interesting pattern occurs for the prediction of Equinor, in which the model makes fairly accurate predictions up to a certain point in which it follows the opposite pattern of the actual prices until it slowly progresses into static values. This effect could indicate that the model has learned the historical patterns well but struggles to capture the dynamics of the data beyond that point, most likely due to the shift in prices. This phenomenon is commonly referred to as "concept drift" or "data drift" in time series forecasting. Concept drift occurs when the underlying patterns or relationships in the data change over time, making the previously learned patterns less relevant or informative for future predictions. In such cases, the model may struggle to adapt to the new patterns and continue to rely on the historical patterns it has learned (Wang et al., 2011).

One intriguing observation from the RF model's predictions is its behavior in relation to the DNB stock, where it appears to have an upper threshold. Once the actual values surpass approximately 110 NOK, the predictions become stagnant. One possible explanation for this effect is that the distribution of the adjusted closing price undergoes a change as it reaches or exceeds 110 NOK. In other words, the underlying patterns might exhibit a shift in linearity beyond a certain threshold, which the model fails to capture. Alternatively, this behavior could be attributed to the complexity of the model, resulting in an underfit of the data. While Kumar et al. (2018) demonstrated the Random Forest model's effectiveness in explaining complex market dynamics, the current analysis yields different results. However, it is worth noting that the Kumar et al. (2018) emphasized the model's efficacy in larger datasets. Applying the Random Forest model to even larger datasets with more features might yield improved results.

The evaluation metrics indicate that the Random Forest model outperformed the XGB model, despite both models exhibited similar patterns. This discrepancy in performance could be attributed to the models' treatment of outliers and noisy data. It is possible that the Random Forest model displays a higher level of robustness and is less affected by these factors thereby leading to superior results.

The LSTM model’s predictions introduced an interesting finding for the two stocks Mowi and Telenor, where there was a certain “lag” in the predictions. This lag in prediction accuracy may be due to a lack of relevant features. The model seems often to struggle with accurately identifying patterns during periods of peaks and lows. These fluctuations could be caused by external factors not captured by the price data alone. For instance, in the case of Mowi, we observe a significant price drop around March 2020, potentially linked to the impact of the pandemic. Although the model still manages to capture some aspects of the pattern during this period, its performance is not as strong as in other periods. Additionally, this “lag” effect could indicate that rather than using previous data to accurately predict what the adjusted closing price will be, the model simply assumes that the price tomorrow must be the same price as today, further research is however needed to address this.

Additionally, some of the models were able to make fairly accurate predictions, but most models seemed to perform worse during the end of the test period which was the middle of 2021 and 2023. Possible explanations for this effect could be external factors that made sudden unexpected shifts in the market. For example, during 2021, Norway experienced a surge in inflation that was the biggest in 13 years (Knudsen, 2021). When inflation rises, stable stocks or value stocks are often traded more due to their stability (Zucchi, 2023), which could account for a shift in the development of the stock price that the model is not able to capture.

Among the evaluated models, the LSTM model produced the best results, demonstrating relatively accurate predictions compared to the other models. This finding aligns with previous research suggesting that artificial neural networks, such as LSTMs, excel in predicting stock market trends (Galler & Kryzanowski, 1993; Nabipour et al., 2020). Thus, the superiority of the LSTM model in capturing the dynamics of the Norwegian stock market further strengthens the notion that LSTM models are well-suited for this domain.

Overfit

The OLS model produced an interesting finding regarding the DNB stock, in which the model was able to effectively capture the variability of the data. Possible explanations for the unusually strong performance could include overfitting, where the model has learned noise or specific patterns from the training data too well. Additionally, the presence of anomalies or outliers within the dataset may significantly impact the model's performance.

Yadav et al. (2020) emphasized the significance of hyperparameter tuning in addressing specific modeling problems. In the course of the modelling, the LSTM model exhibited signs of overfitting, prompting the exploration of hyperparameters that could enhance accuracy while mitigating overfitting. The optimizer and dropout rate emerged as the most influential hyperparameters for improving model performance and decreasing overfit. Specifically, the adagrad optimizer with a

dropout rate of 0.1 demonstrated superior results when predicting the stocks considered in this study. These findings support Yadav et al.'s (2020) claims regarding the importance of hyperparameter tuning. It is worth mentioning that there might be other hyperparameters with greater combined impact, but due to computational limitations of this thesis, a more comprehensive analysis was not performed. Despite this limitation, the observed effects of optimizing the specified hyperparameters underscore the value of hyperparameter tuning in enhancing model performance.

Efficient Market hypothesis

Based on the results, it is difficult to draw a definitive conclusion regarding the efficiency of the market. The performance of the LSTM model across all stocks yields favorable metric scores, including an R2 score of 0.47, MSE of 95.34, RMSE of 7.29, MAE of 6.62, and MAPE of 10.94%. While these scores do not provide direct evidence of market inefficiencies, the model's ability to generate reasonably accurate results suggests the possibility of existing inefficiencies. The discrepancy between the model's predictions and the actual prices can indicate that there may be exploitable patterns or factors influencing stock movements that are not fully incorporated into the efficient market hypothesis. However, further analysis and examination of additional variables would be necessary to make a more conclusive determination regarding market efficiency.

5.3 Limitations

While prediction models provide valuable insights, they will also be prone to limitations and difficulties. I will discuss some of the limitations with the chosen approach.

5.3.1 Feature Engineering

One limitation of the analysis is the narrow focus on feature engineering, where only the adjusted closing price is considered. While this approach may have its reasons, a more comprehensive feature engineering process could potentially yield better results

and provide deeper insights into the underlying behavior of the market. By incorporating additional relevant features, such as trading volume, technical indicators, or macroeconomic data, the models could capture a broader range of factors influencing stock prices, which could yield more accurate predictions.

5.3.2 Fixed Window Size

The utilization of a fixed window size of 60 for both the training and test data is another limitation. While this approach captures consistent patterns and relationships between historical data and future predictions, it may not be optimal for all stocks. Different stocks might exhibit varying behaviors and require different window sizes to effectively capture relevant patterns. Exploring adaptive or variable window sizes tailored to each stock could enhance the models' ability to uncover meaningful insights specific to each stock's dynamics.

5.3.3 Generalization

One limitation of this thesis is the lack of testing the models' ability to generalize to stocks with lower market capitalization or liquidity. The models were primarily evaluated on four stocks with relatively high market capitalization, which may not adequately represent the broader stock market. It is important to consider that less liquid stocks may exhibit different patterns and behaviors due to lower trading volumes and limited market activity. Therefore, the findings and performance of the models in this thesis may not extend seamlessly to these stocks.

5.3.4 Alternative Models

Although the used models are based on recommendations from the literature, it is important to acknowledge that there are other machine learning models available for stock prediction. The selection of models used in the analyses may limit the exploration of potentially valuable approaches for identifying meaningful patterns. Considering alternative models and comparing their performance could provide a more comprehensive understanding of the predictive capabilities in stock market analysis.

5.4 Future research

This thesis has primarily focused on exploring different machine learning models best suited for the Norwegian stock market. However, future research area should extend the complexity of the models. Possible extensions to the performed analysis could involve incorporating sentiment analysis in combination with convolutional neural networks. By analyzing text from news articles, social media, and other relevant sources of stock discussions, the models could account for trends influenced by individual investors. Additionally, conducting more extensive feature engineering would be beneficial, going beyond mere accuracy improvements, and striving to uncover more nuanced patterns and signals that can be utilized for profitable stock trading.

Furthermore, to enhance the generalization capabilities of the models, it would be important to consider including stocks with varying degrees of liquidity in future research. Incorporating less liquid stocks in the training and evaluation process would allow for a broader assessment of the models' performance across different market conditions. These stocks often exhibit distinct behaviors due to lower trading volumes and limited market activity, providing an opportunity to evaluate the models' robustness and their ability to generalize beyond highly liquid stocks.

Lastly, simulating different trading strategies using the trained models would be an interesting approach to evaluate their practical applicability and assess their performance in real-world scenarios. By simulating various trading scenarios and analyzing the outcomes, researchers can gain insights into the models' effectiveness in generating profitable trading decisions and explore potential avenues for improving their performance.

By addressing these research directions, future studies can advance our understanding of machine learning models' capabilities in stock market analysis, enhance their generalization abilities, and explore their practical applicability in real-world trading environments.

6.0 Conclusion

This thesis aimed to explore the predictive performance of five different models, namely Ordinary Least Squares, Support Vector Regression, Random Forest, Extreme Gradient Boosting, and Long Short-Term Memory networks. The models were assessed on their ability to accurately predict the adjusted closing price of four stocks: Equinor, Mowi, Telenor, and DNB. The analysis utilized a rolling window approach, where each model received 60 days of data for each prediction point, covering the period from 01.01.2001 to 01.01.2023. To ensure robustness and prevent overfitting, extensive hyperparameter tuning and cross-validation techniques were employed.

Evaluation of model performance employed several performance metrics, including $R^2, MSE, RMSE, MAE, and\ MAPE$. Among the models, the Random Forest model demonstrated the best performance for predicting the adjusted closing price of Mowi, while the LSTM model consistently outperformed across all other stocks and achieved the best aggregated results. These findings align with previous research that highlights the effectiveness of LSTM models in stock market prediction. Notably, the models exhibited decreased performance during periods of exogenous shocks, such as the COVID-19 pandemic and spikes in inflation. This suggests that the patterns observed in historical prices alone may not adequately anticipate or account for sudden market disruptions.

In terms of the efficient market hypothesis, this study does not provide a conclusion of an efficient or inefficient market. However, the prediction performance of the LSTM model implies that there are potential market inefficiencies that could be exploited. Further analysis and investigation of additional variables are necessary to draw more definitive conclusions regarding market efficiency.

In summary, this study demonstrates that the LSTM model offers the highest prediction accuracy for stocks listed on the Norwegian stock market. The findings contribute to the existing literature on stock prediction in various markets and offer

insights into potential market inefficiencies. Future research can build upon these findings to further enhance prediction accuracy in subsequent models.

Bibliography

Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. Journal of Finance, 25(2), 383-417.

Fama, E. F., Fisher, L., Jensen, M. C., & Roll, R. (1969). The Adjustment of Stock Prices to New Information. International Economic Review, 10(1).

Naseer, M., & Tariq, Y. B. (2015). The Efficient Market Hypothesis: A Critical Review of the Literature. The IUP Journal of Financial Risk Management, 12(4), 48-63.

Konak, F., & Seker, Y. (2014). The Efficiency of Developed Markets: Empirical Evidence from FTSE 100. Journal of Advanced Management Science, 2(1).

Malkiel, B. (2003). The efficient market hypothesis and its critics. Princeton University, CEPS Working Paper No. 91.

Ball, R., & Brown, P. (1968). An empirical evaluation of accounting income numbers. Journal of Accounting Research, 6(2), 159-178.

Bernard, V. L., & Thomas, J. K. (1990). Evidence that stock prices do not fully reflect the implications of current earnings for future earnings. Journal of Accounting and Economics, 13, 305-340.

Chowdhury, M., Howe, J. S., & Lin, J. (1993). The relation between aggregate insider transaction and stock market returns. The Journal of Financial and Quantitative Analysis, 28(3).

Pettit, R. R., & Venkatesh, P. C. (1995). Insider Trading and Long-Run Return Performance. Financial Management, 24.

Kryzanowski, L., Galler, M., & Wright, D. W. (1993, 7). Using Artificial Neural Networks to Pick Stocks. Financial Analysts Journal

Kumar, I., Dogra, K., Utreja, C., & Yadav, P. (2018). A comparative study of supervised machine learning algorithms for stock market trend prediction. In Proceedings of the 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT 2018) (pp. 1003-1007). Coimbatore, India: IEEE. doi: 10.1109/ICICCT.2018.8473214.

Nabipour, M., Nayyeri, P., Jabani, H., S. S., & Mosavi, A. (2020). Predicting Stock Market Trends Using Machine Learning and Deep Learning Algorithms Via Continuous and Binary Data; a Comparative Analysis. IEEE Access, 8, 150199-150212. doi: 10.1109/ACCESS.2020.3015966.

Yadav, A., Jha, C. K., & Sharan, A. (2020). Optimizing LSTM for time series prediction in Indian stock market. *Procedia Computer Science*, *167*, 2091–2100. https://doi.org/10.1016/j.procs.2020.03.257

Becker, S. (2021, December 28). The efficient market hypothesis posits that all stocks are fairly valued, making it virtually impossible to earn big profits. Business Insider. Retrieved from https://www.businessinsider.com/personal-finance/efficient-market-hypothesis?r=US&IR=T

Smith, T. (2023). Random Walk Theory: Definition, How It's Used, and Example. Investopedia. Retrieved from https://www.investopedia.com/terms/r/randomwalktheory.asp

Singal, V. (2003). Beyond the Random Walk: A Guide to Stock Market Anomalies and Low-Risk Investing. Oxford University Press.

Levy, R. A. (1967 The Theory of Random Walks: A Survey of Findings. The American Economist.

Russell, S. J., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach. Prentice Hall.

Géron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (2nd ed.)*. O'Reilly Media, Inc.

Kavitha, S., Varuna, S., & Ramya, R. (2016). A Comparative Analysis on Linear Regression and Support Vector Regression. IEEE.

Burkov, A. (2019) The Hundred-Page Machine Learning Book. 159.

Aggarwal, C. C. (2018). Neural Networks and Deep Learning: A Textbook. Springer International Publishing AG.

Chollet, F. (2017). Deep Learning with Python. Manning Shelter Island.

Joseph, M. (2022). Modern Time Series Forecasting with Python: Explore industry-ready time series forecasting using modern machine learning and deep learning. Packt Publishing.

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice*. (2nd ed.) OTexts.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning: With Applications in R (Vol. 112). Springer.

Awad, M., & Khanna, R. (2015). Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers. Apress Berkeley, CA. Retrieved from https://doi.org/10.1007/978-1-4302-5990-9

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Stanford, CA: Stanford University.

Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794). ACM.

Kuhn, M., & Johnson, K. (2013). Applied Predictive Modeling. Springer.

scikit-learn (n.d. a). *Sklearn.Linear_model.LinearRegression*. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

scikit-learn (n.d. b). *Sklearn.Svm.SVR*. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR

scikit-learn (n.d. c). *Sklearn.Ensemble.RandomForestRegressor*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Keras (n.d.). *The Sequential model*. Keras. https://keras.io/guides/sequential_model/

Scott, G. (2022, April 29). Liquid Market: Definition, Benefits in Trading, and Examples. Investopedia. Retrieved from https://www.investopedia.com/terms/l/liquidmarket.asp

Wang, S., Schlobach, S., & Klein, M. (2011). Concept drift and how to identify it. Journal of Web Semantics, 9(3), 247-265. https://doi.org/10.1016/j.websem.2011.05.003

Knudsen, C. (2021). Inflasjonen stiger til det høyeste på 13 år. e24. Retrieved from https://e24.no/norsk-oekonomi/i/RrE2aA/inflasjonen-stiger-til-det-hoeyeste-paa-13-aar

Zucchi, K. (2023). Inflation's Impact on Stock Returns. Investopedia. Retrieved from https://www.investopedia.com/articles/investing/052913/inflations-impact-stock-returns.asp

Ganti, A. (2020, December 28). *Adjusted Closing Price*. https://www.investopedia.com/terms/a/adjusted_closing_price.asp