Phi Thien Hoang
Jonas Laskemoen

Predicting stock prices with Long Short-Term Memory based models using a combination of data sources

Master's thesis in Computer Science
Supervisor: Prof. Björn Gambäck
August 2020

Phi Thien Hoang
Jonas Laskemoen

Predicting stock prices with Long Short-Term Memory based models using a combination of data sources

Master's thesis in Computer Science
Supervisor: Prof. Björn Gambäck
August 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

The focus of this thesis is stock price prediction using easily available sources of information. The four research questions presented are related to identifying patterns in the gathered data (1), comparing Long Short-Term Memory (LSTM) based models to simpler baseline models (2), analyzing the effect of introducing a novel context module to the LSTM based models (3) and analyzing the effects of generalizing models (4). Three groups of data were used, representing trading data, sentiment data and trendscore data. The performance of the models were measured in terms of mean absolute percentage error (MAPE), mean absolute error (MAE), mean squared error (MSE) and direction accuracy (DA). Generally, the LSTM based models were inferior to the baseline models and seemed to converge to the naive 1-step-behind model, a model that always predicts the next price to be the current price. However, one LSTM configuration managed to improve statistically significant over a random guessing model in terms of DA, although only on the time frame related to the test set. The inconsistencies across time frames led to the conclusion that the model did not seem suitable for practical use. Multiple hypotheses explaining why this task is as hard as witnessed are presented, mainly related to the varying properties across time frames; symbolized by the price variances, differences across stocks, and the amount of usable information in the available data.

Sammendrag

Fokuset for denne oppgaven er prediksjon av aksjekurser ved hjelp av lett tilgjengelige informasjonskilder. De fire presenterte forskningsspørsmålene er relatert til å identifisere mønstre i de innsamlede dataene (1), sammenlikne Long Short-Term Memory (LSTM) -baserte modeller med enklere grunnmodeller (2), analysere effekten av å introdusere en ny kontekstmodul til de LSTM-baserte modellene (3) og analysere effekten av generaliserende modeller (4). Tre grupper av data ble brukt, som representerer handelsdata, sentimentdata og trendscore-data. Ytelsen til modellene ble målt i form av gjennomsnittlig absolutt prosentvis feil (MAPE), gjennomsnittlig absolutt feil (MAE), gjennomsnittlig kvadratfeil (MSE) og retningsnøyaktighet (DA). Generelt var de LSTM-baserte modellene dårligere enn grunnmodellene og syntes å konvergere til en modell som alltid forutsier at neste pris skal være den nåværende prisen. Imidlertid klarte en LSTM-konfigurasjon å forbedre seg statistisk signifikant over en tilfeldig gjetningsmodell når det gjelder DA, men bare på tidsrammen relatert til testsettet. Uoverensstemmelsene på tvers av tidsrammene førte til konklusjonen at modellen ikke syntes egnet for praktisk bruk. Flere hypoteser som forklarer hvorfor denne oppgaven er så vanskelig som observert presenteres, hovedsakelig knyttet til de forskjellige egenskapene på tvers av tidsrammene; symbolisert av prisavvik, forskjeller mellom aksjer og mengden brukbar informasjon i tilgjengelige data.

Preface

We started this project with a lot of interest in the stock market, but had little to no practical experience with it. Coming from a computer science background with specialization in artificial intelligence, we wanted to see what computers would be able to do in this field as of now. Working on this project has given us better understanding of the stock market, as well as deeper knowledge of practical usage of machine learning tools, that we hope could be useful in our future endeavours. We hope that the readers are at least as fascinated by our findings as we initially were.

We would like to thank our supervisor Professor Björn Gambäck for his guidance. All the help, advice and suggestions have been greatly appreciated and played an integral part in the finished thesis. His willingness to accept the task of being our supervisor despite his tight schedule gave us the extra motivation needed to complete this project.

Contents

List of Figures

List of Tables

1 Introduction

The efficient market hypothesis is a widely known hypothesis related to stock trading (Fama, 1970). The hypothesis posits that the market is perfectly efficient in reflecting all public information. When news appear, the prices in the market will immediately and perfectly reflect these news (Malkiel, 2003). Price movements will only reflect previously unknown information, i.e. news. As news are unpredictable, price movement therefore follows a random walk. This implies that expert stock traders would be no more efficient in earning than amateurs, which has been shown to not be the case. According to Investopedia.com, 2020, the Windsor stock fund had a yearly return of 13.7% vs 10.6% for the S&P500 from 1964 to 1995, which adds up to a return of 53 times the initial investment over 31 years. This shows that there is an enormous potential for making money in the stock market for expert traders. In later years, the sentiment has moved from the market being perfectly efficient to the notion that it is extremely efficient. The market does not move according to a perfectly random walk and previous information can be used to predict future price movements. As the market is extremely efficient in reflecting public information, one way of consistently beating the market is to find superior predictive patterns in this information that few others find.

Being efficient in identifying complex patterns is a fundamental property of humans, this is how we make sense of the world around us. Generally we are extremely efficient in identifying these patterns as a result of the refinement process of natural selection. Solving hard pattern recognition tasks has therefore historically been assigned to humans, but in later years computers have due to an increase of available data, optimization of hardware, and the emergence of machine learning algorithms surpassed humans in hard, abstract problems. According to Scientificamerican.com (2020) even in facial recognition Artificial Intelligence (AI) has reached the level of the best forensic examiners, i.e. top human experts. AI can therefore be a valid tool for pattern recognition tasks. These results combined with the massive amounts of information related to the stock market is a strong indicator that a machine learning algorithm is an interesting candidate for stock prediction. The thesis will therefore explore the efficacy of using machine learning algorithms for predicting future stock prices.

The overarching goal of this thesis can be summarized as following: **Investigate whether easily available data can be used to accurately predict stock prices using machine learning**. To limit the scope and make the task more manageable, the thesis will focus on achieving this goal by answering the following research questions:

I) **Are there some patterns in the trading data, sentiment data and search popularity data gathered in this project that can facilitate price prediction?**

II) **Will a model based on LSTMs outperform the baseline models in predicting the next day prices of stocks?**

III) **Will introducing a context module improve prediction?**

IV) **Can one of the models with configurable parameters in this project outperform the baseline models in predicting stock prices, using the same set of parameters for every stock in this project?**

Motivation for the research questions mentioned above will be explained in the following paragraphs.

In later years, the number of users on social media has increased dramatically. According to Statista.com (2020) Facebook had 2.375 billion monthly active users in the first quarter of 2019. Twitter had 326 million monthly active users by 26th of October 2018. These services are platforms in the sense that users can express their own opinion for other users to digest. Information flow between humans is now, more than ever, facilitated through these platforms. People express their opinions on virtually everything, ranging from individual opinions on sports and family life, to covering wider issues such as politics and stock markets. In this plethora of information, is there anything that can be useful for finding some predictive patterns related to the stock market? Bollen, Mao & Zeng (2011) measured the general mood given Twitter posts in order to improve stock market predictions with positive results. In addition to social media, there are many other services where a large proportion of the world express themselves. Google, the most widely used search engine is one such example. On this service, users express their own curiosities by searching. In 2012, over 1.2 trillion searches were conducted through Google (Internetlivestats.com, 2019). Preis, Moat & Stanley (2013) found some correlation between Google search trends and market price movements, and concluded that Google search trends might even provide insight into future market trends. Research question I) was motivated by the possibilities present in such data. The machine learning models proposed in this thesis will therefore use Twitter and Google trends data in order to predict the stock market, in addition to historical trading data. Twitter data in the form of historical sentiment scores produced by sentiment analysis by the service StockFluence (2020), and Google data in the form of historical search popularity data.

Research question II) was derived from the preliminary research. The results found in the preliminary research on related works, such as the works of Althelaya, El-Alfy & Mohammed (2018), Jiahong Li, Bu & Wu (2017) and Xiong, Nichols & Shen (2016) show that Long Short Term Memory neural network is one of the models that is often considered when working with sequential data such as a time series, and consistently delivers decent performance in most of the works, even achieving state-of-the-art performance in some cases. This thesis will therefore use and analyse LSTM models in order to find predictive patterns in the data. These models will be compared to simpler models that will act as the baseline models, such as a linear regression model, a ridge regression model, and a simple model that always predicts the price of tomorrow to be equal to the price of today.

Different configurations of LSTMs will be examined in the thesis, all implemented using Keras (2020) with TensorFlow (2020) backend. A **vanilla LSTM model**, an LSTM model which no modification is done will be presented. A **stacked LSTM**, meaning a model with more than one LSTM layer will also be presented, as adding layers often means increasing the ability to capture complex non-linear relationships. Since this task is often regarded as one of complex nature, investigating whether adding more layers could combat this complexity could be of value. To investigate whether improvements could be made when analysing the data both forwards and backwards, a **bidirectional LSTM** model was also implemented. According to Althelaya, El-Alfy & Mohammed (2018), a tuned (optimized number of hidden units and number of epochs) bidirectional LSTM model performed better on predicting stock prices given historical prices compared to both an untuned simple LSTM model and a tuned stacked LSTM model.

Additionally, experiments on a configuration trying to provide context to the model will be conducted. This is done by making the model trying to specialize in what stock it is learning on, by providing meta-information such as the name of the stock it is currently training on. This configuration is inspired by the encoder-decoder architecture mentioned in Goodfellow, Bengio & Courville (2016). The usage of information to guide the model on what to focus on is the main inspiration of the configuration trying to utilize context information, and the reason why research question III) is one of the main focuses of this thesis.

The last thing this project has focused on is whether it is possible to find a model and associated configurations that work well on a set of different stocks, without the necessity of manually fine-tuning the configurable parameters for each and every stock. This is equivalent to answering research question IV), which was motivated by the amount of resources needed to manually fine-tune models, and amount of configurable parameters a machine learning model may have. This was also motivated by the fact that manually fine-tuning models for each stock makes the model less flexible in practical usage, and makes it harder to make use of information that can be shared across multiple stocks, if any exists at all.

Contributions to this field made by this thesis are multiple. First of all, the thesis contributes to the field with comparisons between the LSTM based models and a simpler model that constantly lags one step behind the actual values with some key observations. In terms of experimentation with new implementation of models, this thesis also contributes by presenting and analyzing a novel context module that is added to the LSTM models. Another contribution is the investigation and evaluation of results not only on one time frame, but on two consecutive time frames in order to present a more comprehensive investigation of the models. The last contribution of this thesis is illustrating the challenges related to this task by presenting and discussing them.

This thesis starts off with the fundamental understanding necessary in Chapter 2, in addition to a brief introduction of the main tools used. Chapter 3 presents the preliminary research on related works. Chapter 4 provides insights on the data utilized in this project, as well as the data sources and any preprocessing deemed necessary. In Chapter 5, the architecture and design of the implemented models are presented and described. A thorough description of the experimental setup, including implementations and data partitioning, is provided in Chapter 6. Following this, Chapter 7 details how the experiments were organized and carried out. Results obtained from the experiments are presented in Chapter 8, along with brief discussions of some interesting observations and highlights. In Chapter 9, the results are discussed and compared. Chapter 10 draws the conclusion, and Chapter 11 presents possible future works in this field.

2 Background

This chapter provides background knowledge necessary for this project, divided into essentials and additional knowledge, to provide insight on what is directly related to the core parts of this thesis. The information is presented to give understanding of the subjects and ideas this project is based upon, as well as techniques that need to be implemented in the experimental part. Some methods for data analysis and evaluation of the results are also presented. Lastly, the final section of this chapter gives a brief description of the tools utilized in this project.

2.1 Essentials

The knowledge deemed necessary to understand the core parts of the project will be presented here. This includes theoretical knowledge of the models, metrics, and other associated concepts that are directly related to the project.

2.1.1 Sentiment analysis

The sentiment of a text can be viewed as the reduction of the text into different categories by extracting the opinion and subjectivity of the text. Categories often used are “positive”, “negative” and “neutral”, but dividing text into categories are not limited to these only. Other examples of categories are different emotions, for instance “anger”, “sadness”, “fear” and “happiness”,

Humans are social and emotional beings and we write texts that express emotions. Consider the text “This is one of the best days of my life”. It is easy to see that this text expresses emotions such as ecstasy and happiness which in turn clearly are positive emotions - this text can therefore be viewed as having a positive sentiment. It is important to have in mind that it is not always that simple. Sometimes the text might express conflicting emotions - not all texts are either positive or negative but might express strong positive and strong negative emotions. Also, considering the possibility of sarcasm, properly identifying the sentiment can prove challenging.

Sentiments give us insight into how people interpret the world around them. People act on their interpretations of the world, and in that way, sentiments can help us in predicting behavior. For instance, if a person expresses highly negative emotions towards a presidential candidate, chances are that the person is not going to vote for that candidate.

Sentiment analysis is the act of extracting the sentiment of texts, usually the act of making machines able to extract this information automatically. Sentiments extracted are often used to give further insight on some task or issue, for instance the task of finding the most popular presidential candidate or predicting the winner of an election.

2.1.2 Prediction

Prediction is the act of estimating a future value or providing a statement about a future event, using available and preferably relevant information. Prediction is a major focus in machine learning and is employed in various fields, such as financial markets to predict values or growth of stocks, or in sports to predict results of matches.

Within the field of machine learning, prediction is done by providing a machine learning model input data, often divided into a set of instances $I = (i_1, i_2, i_3, \ldots, i_n)$ of which each contains a set of features $X = (x_1, x_2, x_3, \ldots, x_n)$. The task is then to predict future instances. In supervised learning, each instance also contains a label y, which can be used to find the way of dividing into different classes, also known as classification.

The act of prediction can also be extended to predict continuous values instead of a defined set of limited classes. This is then called regression, and is done by having the machine learning model learn a way to use X to find an approximation as close as possible to the true value y.

2.1.3 Time series

Time series are data sets that consist of a sequence of data that are indexed and ordered by time, often with a constant time interval between the instances of the sequence. Time series can be divided into different types:

- Linear and non-linear time series: A linear time series is a time series where each data point can be viewed as a linear combination of past data points. A non-linear time series is a time series that is not linear.
- Deterministic and non-deterministic time series: Time series that are deterministic tend to follow a set of rules, making the time series behave in a certain way. Non-deterministic time series, on the other hand, exhibit stochastic or random behavior.

2.1.4 Supervised learning

Supervised learning is approximation of a function f using pairs of input and outputs of the function: $\{X_1, f(X_1)\}, \{X_2, f(X_2)\}, ..., \{X_n, f(X_n)\}$.

In practice, supervised learning is used in many applications, such as object classification of images, regression tasks, etc. In recent years, neural networks have seen massive adoption as they are effective for applications that contain large amounts of noisy data.

2.1.5 Overfitting

Overfitting can be summarized as the act of a supervised learning model being overly specialized in prediction on the trained data so that the performance on predicting unseen data is low, meaning the supervised learning model is bad at generalization.

Let X, Y be the set of all possible input data and output data, respectively, and let f be a function so that $f : X \rightarrow Y$. Then let $X_{observed}$ be the set of input data that the supervised learning model can be trained upon, and X_{unseen} be the set of input data that has not been seen. We have that $X_{observed} + X_{unseen} = X$. Let $\hat{f}$ be the approximation function learned from training on $X_{observed}$, so that $\hat{f} : X_{observed} \rightarrow Y_{observed}$ performs well, meaning that $\hat{f}$ performs close to or as good as f on this task. If the model is overfitted, then $\hat{f} : X_{unseen} \rightarrow Y_{unseen}$ will not perform well, even though $\hat{f} : X_{observed} \rightarrow Y_{observed}$ performs well.

2.1.6 Linear Regression

Linear regression is a statistical method used to investigate the linear relationship between a dependent variable and independent variables. Dealing with the relationship between a dependent variable and multiple independent variables is often referred to as multiple linear regression. In machine learning we can express multiple linear regression as a model that assumes a relationship between y, the variable to be predicted, and p input features $x_1, x_2, x_3, \ldots, x_m$:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_m x_m + \epsilon, \tag{1}$$

where $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_m$ are regression coefficients and ϵ is the random error of the model (Yan & Su, 2009).

Given a data set which consists of n sets of variables to be predicted and input features,

$\{y_i, x_{i1}, x_{i2}, \ldots, x_{ip}\}_{i=1}^{n}$, and we have that

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1p} \\ 1 & x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{np} \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix} \text{ and } \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}.$$

the linear relationship can be represented as:

$$\mathbf{y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon}. \tag{2}$$

y_i is the variable to be predicted of the ith set, x_{ij} is the jth input feature of the ith set, and ϵ_i is the random error of the ith set.

Building the multiple regression model requires the regression coefficients $\beta_0, \beta_1, \beta_2, \ldots, \beta_m$ to be estimated. Finding the least square solution is a frequently used method. This means finding the coefficients that minimizes $\sum_{i=1}^{n} \epsilon_i$. Using this method, the estimated regression coefficients $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_m$ can then be defined as:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\mathbf{y}. \tag{3}$$

2.1.7 Ridge regression

Ridge regression can be viewed as a variation or an extension of linear regression mentioned in Section 2.1.6. It includes $\mathbf{L}^2$ regularization, which makes the model act as if the input features $X = (x_1, x_2, \ldots, x_p)$ have a higher variance, making it shrink the corresponding set of β values of features that have a lower covariance with the output compared to this added variance from the regularization (Goodfellow, Bengio & Courville, 2016, p. 231-234). The estimated regression coefficients is then defined as:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\mathbf{y}, \tag{4}$$

where the added $\lambda\boldsymbol{I}$ term represents the extension of a linear regression model that uses a least square solution, such as the one explained in Section 2.1.6.

2.1.8 Artificial neural networks

An artificial neural network (ANN) is a mathematical model represented by a weighted directed graph. This model defines a function $\hat{\boldsymbol{f}}$ that is used as a function approximator of a specific function $\boldsymbol{f}$. Generally, ANNs are used for supervised learning problems, i.e. approximating the function $\boldsymbol{f}$ using input/output pairs
$\{(\boldsymbol{x_0}, \boldsymbol{f}(\boldsymbol{x_0})), (\boldsymbol{x_1}, \boldsymbol{f}(\boldsymbol{x_1})), ..., (\boldsymbol{x_n}, \boldsymbol{f}(\boldsymbol{x_n}))\}$.

This section will present an illustration of how a neural networks work in a very simplified and general way. There are many ways to implement a neural network, but the focus will be on the most salient concepts needed to be understood at a high level on how a neural network works. Vector notation will be used, i.e. representation of vectors by lowercase letters in bold, e.g. $\boldsymbol{x}$. Matrices are represented by uppercase letters in bold, e.g. $\boldsymbol{W}$. Also, functions that project a vector onto a vector space will be represented by lowercase letters in bold, e.g. $\boldsymbol{\sigma} : \mathbb{R}^m \rightarrow \mathbb{R}^n$

The graph

To understand the neural network, one can first look at the building block of the graph - the perceptron, seen in Figure 1. The perceptron is a representation of a function that transforms an input signal $\boldsymbol{x} = [x_0, x_1, ..., x_n]$ received from preceding nodes through some edges $\boldsymbol{w} = [w_0, w_1, ..., w_n]$ into a value a. The perceptron combines $\boldsymbol{x}$ and $\boldsymbol{w}$ in a function σ to produce the output a. To make the perceptron able to represent nonlinear functions, σ must be a nonlinear function.

$a = \sigma(\boldsymbol{x} \cdot \boldsymbol{w}^T)$

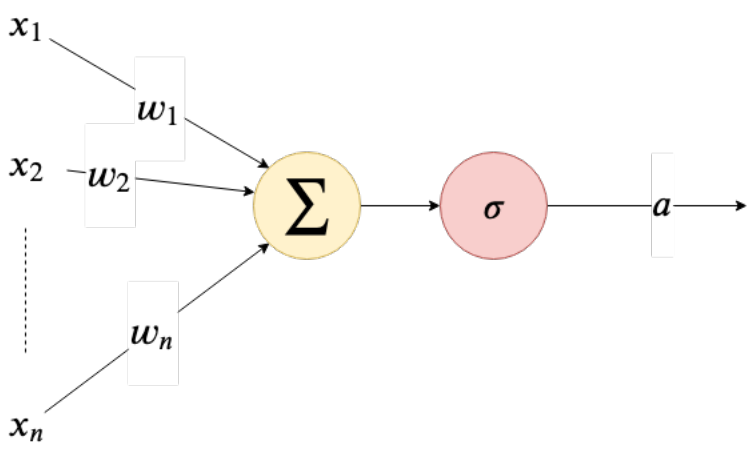


Figure 1: Visualization of the perceptron

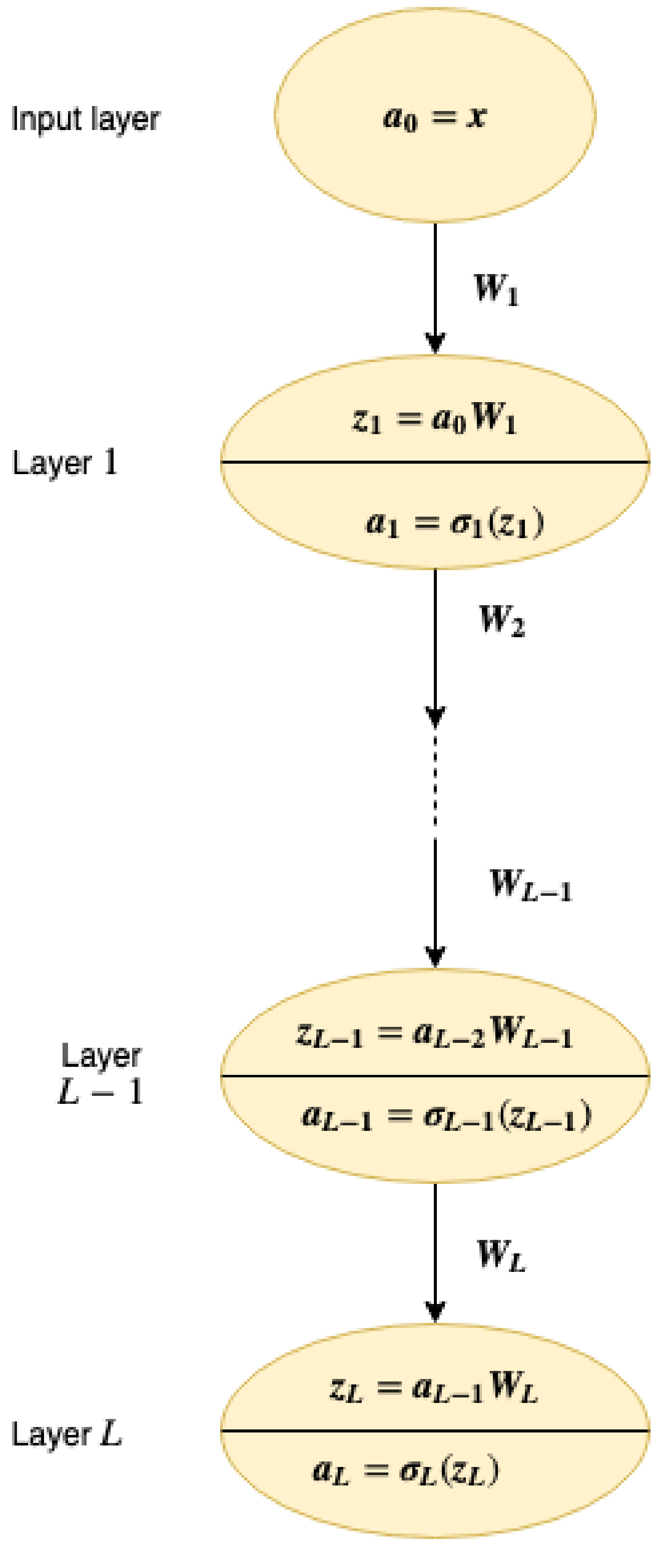


Figure 2: Visualization of the neural network $\hat{f}$, layer by layer

In the network, perceptrons are arranged in layers. A neural network is shown in Figure 2. Calculations can be done by calculating the output of a complete layer by using matrix multiplication. E.g. the output of the second layer is $\boldsymbol{a_1} = \boldsymbol{\sigma_1}(\boldsymbol{a_0} \cdot \boldsymbol{W_1})$, where $\boldsymbol{\sigma_1}$ is a function that transforms a vector into a vector of the same dimension, $\boldsymbol{\sigma} : \mathbb{R}^n \Rightarrow \mathbb{R}^n$. The weights between two succeeding layers with j and k amount of nodes respectively are represented by the weight matrix:

$$\boldsymbol{W}^i = \begin{bmatrix} w^i_{0,0} & w^i_{1,0} & \cdots & w^i_{j,0} \\ w^i_{0,1} & \ddots & & \\ \vdots & & & \\ w^i_{0,k} & & & w^i_{j,k} \end{bmatrix}$$

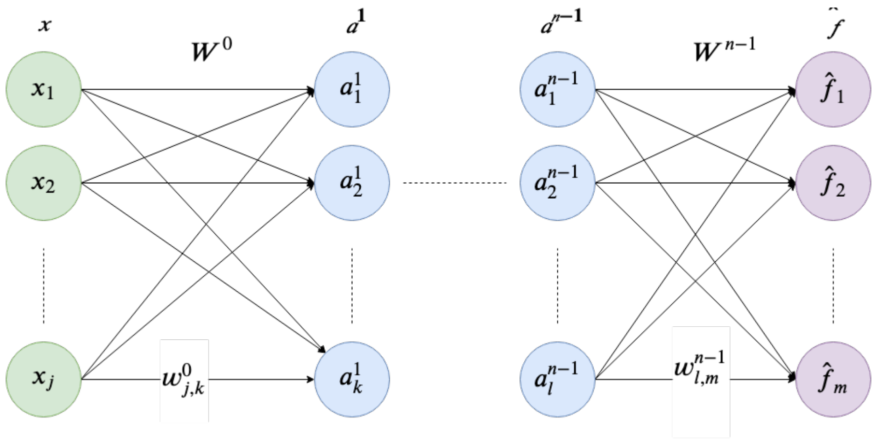


Figure 3: Visualization of an n-layered neural network transforming $\boldsymbol{x}$ into $\hat{\boldsymbol{f}}$

Learning

Let the function $\boldsymbol{f}$ be the one to be approximated with a neural network $\hat{\boldsymbol{f}}$. First step is to initialize the network $\hat{\boldsymbol{f}}$ with random weights. Since the goal is to have $\hat{\boldsymbol{f}}$ approximate $\boldsymbol{f}$, $\hat{\boldsymbol{f}}(\boldsymbol{x})$ should be as close as possible to $\boldsymbol{f}(\boldsymbol{x})$ for all $\boldsymbol{x}$. To do this, one must define the distance function (the network needs to know what outputs are good and which are bad), also called the loss function, $L(\hat{\boldsymbol{f}}, \boldsymbol{f}, \boldsymbol{x})$. Then, to minimize this loss, modifications of the weights in the network are needed. This is done by using partial derivatives to incrementally improve the network, therefore the loss function must be differentiable with respect to the weights in the network. The method for minimizing the loss in neural networks is generally inspired by the gradient descent algorithm.

Gradient descent is an incremental method to locate local minima in a differentiable function. This method is analogous to walking to a local bottom in a varied landscape. If one simply move in the direction that have the steepest fall, one is guaranteed to eventually reach a local bottom. In the same way, in gradient descent, one can start off at some value $L(x_1)$. Then, finding the gradient can be done by derivating L in terms of all of the parameters (weights in the neural network). Afterwards, one can move in the opposite direction of the gradient to move in the direction of the steepest fall. This is done until the gradient is below a threshold which means being as close to a local bottom as was desired.

To find these gradients the backpropagation algorithm can be of use. Assume an arbitrarily large neural network. Can a formula to find the derivative of the loss in terms of every weight in the network be found? The mathematical operations can be visualized:

To produce an output, follow the graph in figure 2 from top to bottom. First calculate $\boldsymbol{z_1}$, then $\boldsymbol{a_1}$, then $\boldsymbol{z_2}$ etc, all the way to $\boldsymbol{a_L}$ which is the output of the network $\boldsymbol{\hat{f}}$ for an input vector $\boldsymbol{x}$, where $\boldsymbol{\hat{f}} : \mathbb{R}^m \to \mathbb{R}^n$. The goal is to find the gradient, i.e.

$$\triangle L = \left[\frac{\partial L}{\partial \boldsymbol{W_1}}, \frac{\partial L}{\partial \boldsymbol{W_2}}, \ldots, \frac{\partial L}{\partial \boldsymbol{W_L}}\right]$$

Start with finding the derivative of the loss in terms of the final layer, and go backwards through the network until the first layer. To do this, the chain rule which states that

$$\frac{\partial x}{\partial z} = \frac{\partial x}{\partial y_1}\frac{\partial y_1}{\partial y_2}\cdots\frac{\partial y_{n-1}}{\partial y_n}\frac{\partial y_n}{\partial z}$$

for any $x, y_1, y_2, ..., y_n, z$ is needed.

$$\frac{\partial L}{\partial \boldsymbol{W_L}} = \underbrace{\frac{\partial L}{\partial \boldsymbol{a_L}}\frac{\partial \boldsymbol{a_L}}{\partial \boldsymbol{z_L}}}_{\delta_L}\frac{\partial \boldsymbol{z_L}}{\partial \boldsymbol{W_L}} = \delta_L \frac{\partial \boldsymbol{z_L}}{\partial \boldsymbol{W_L}} \qquad \text{Chain rule}$$

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{W_{L-1}}} &= \underbrace{\frac{\partial L}{\partial \boldsymbol{a_L}}\frac{\partial \boldsymbol{a_L}}{\partial \boldsymbol{z_L}}}_{\delta_L}\frac{\partial \boldsymbol{z_L}}{\partial \boldsymbol{a_{L-1}}}\frac{\partial \boldsymbol{a_{L-1}}}{\partial \boldsymbol{z_{L-1}}}\frac{\partial \boldsymbol{z_{L-1}}}{\partial \boldsymbol{W_{L-1}}}\\ &= \underbrace{\delta_L \frac{\partial \boldsymbol{z_L}}{\partial \boldsymbol{a_{L-1}}}\frac{\partial \boldsymbol{a_{L-1}}}{\partial \boldsymbol{z_{L-1}}}}_{\delta_{L-1}}\frac{\partial \boldsymbol{z_{L-1}}}{\partial \boldsymbol{W_{L-1}}} = \delta_{L-1}\frac{\partial \boldsymbol{z_{L-1}}}{\partial \boldsymbol{W_{L-1}}}\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{W_{L-2}}} &= \underbrace{\frac{\partial L}{\partial \boldsymbol{a_L}}\frac{\partial \boldsymbol{a_L}}{\partial \boldsymbol{z_L}}\frac{\partial \boldsymbol{z_L}}{\partial \boldsymbol{a_{L-1}}}\frac{\partial \boldsymbol{a_{L-1}}}{\partial \boldsymbol{z_{L-1}}}}_{\delta_{L-1}}\frac{\partial \boldsymbol{z_{L-1}}}{\partial \boldsymbol{a_{L-2}}}\frac{\partial \boldsymbol{a_{L-2}}}{\partial \boldsymbol{z_{L-2}}}\frac{\partial \boldsymbol{z_{L-2}}}{\partial \boldsymbol{W_{L-2}}}\\ &= \underbrace{\delta_{L-1}\frac{\partial \boldsymbol{z_{L-1}}}{\partial \boldsymbol{a_{L-2}}}\frac{\partial \boldsymbol{a_{L-2}}}{\partial \boldsymbol{z_{L-2}}}}_{\delta_{L-2}}\frac{\partial \boldsymbol{z_{L-2}}}{\partial \boldsymbol{W_{L-2}}} = \delta_{L-2}\frac{\partial \boldsymbol{z_{L-2}}}{\partial \boldsymbol{W_{L-2}}}\\ &\vdots\end{aligned}$$

$$\frac{\partial L}{\partial \boldsymbol{W_1}} = \delta_1 \frac{\partial \boldsymbol{z_1}}{\partial \boldsymbol{W_1}}$$

From the above equations, one can identify a recursive pattern of the derivative of the loss with respect to any arbitrary weight $\boldsymbol{W}_\gamma$ for $\gamma \in \{1, 2, \ldots, L\}$:

$$\frac{\partial L}{\partial \boldsymbol{W}_\gamma} = \delta_\gamma \frac{\partial \boldsymbol{z}_\gamma}{\partial \boldsymbol{W}_\gamma} = \delta_\gamma \frac{\partial}{\partial \boldsymbol{W}_\gamma} \boldsymbol{a}_{\gamma-1} \boldsymbol{W}_\gamma = \boxed{\delta_\gamma \boldsymbol{a}_{\gamma-1}} \tag{5}$$

$$\delta_\gamma = \delta_{\gamma+1} \frac{\partial \boldsymbol{z}_{\gamma+1}}{\partial \boldsymbol{a}_\gamma} \frac{\partial \boldsymbol{a}_\gamma}{\partial \boldsymbol{z}_\gamma} = \delta_{\gamma+1} \frac{\partial \boldsymbol{a}_\gamma \boldsymbol{W}_{\gamma+1}}{\partial \boldsymbol{a}_\gamma} \frac{\partial \boldsymbol{\sigma}_\gamma(\boldsymbol{z}_\gamma)}{\partial \boldsymbol{z}_\gamma} = \boxed{\delta_{\gamma+1} \boldsymbol{W}_{\gamma+1} \boldsymbol{\sigma}'_\gamma(\boldsymbol{z}_\gamma)} \tag{6}$$

$$\delta_L = \frac{\partial L}{\partial \boldsymbol{a}_L} \frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{z}_L} = \frac{\partial L}{\partial \boldsymbol{a}_L} \frac{\partial \boldsymbol{\sigma}_L(z_L)}{\partial \boldsymbol{z}_L} = \boxed{\frac{\partial L}{\partial \boldsymbol{a}_L} \boldsymbol{\sigma}'_L(\boldsymbol{z}_L)} \tag{7}$$

Let it exist a pair $\{\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x})\}$ and that $\hat{\boldsymbol{f}}$ approximating $\boldsymbol{f}$ is wanted. The procedure can be divided into three parts:

1. Forward propagation step: Feed in $\boldsymbol{x}$ into the neural network, producing $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_L\}$.

2. Backward propagation step: Calculate the partial derivatives of the loss in terms of the weights in the last layer, using formulas 5, 6 and 7. Then calculate for the previous layer, etc. until the partial derivatives for all weights is acquired, using the outputs produced in step 1.

3. Gradient descent step: Then update all the weights in the network using the gradient descent method:

$$\boldsymbol{W}_\gamma \leftarrow \boldsymbol{W}_\gamma - \alpha \frac{\partial L}{\partial \boldsymbol{W}_\gamma}$$

where α is the step size

2.1.9 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN), developed based on the works of Rumelhart, Geoffrey E. Hinton & Williams (1986), is a class of ANN that has loops which allows information to persist, making it able to emulate temporal dynamic behavior. This ability makes RNNs especially interesting when dealing with sequential data, where some kind of memory between the sequential information can be important to achieve proper results. This makes RNN models a viable option when dealing with tasks such as sentiment analysis and forecasting, since information from earlier time steps can be useful on these tasks. Other examples of application include handwriting recognition and speech recognition.

Intuition

Let there be a time series, $\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_T}]$, to be transformed in some way. E.g. there exists an array of images of letters to be converted into the word they form. A regular neural network would transform every letter independently of the other letters. Can better results be achieved if the input images are iterated through sequentially and the network is provided with a context which consists of some transformation of previous inputs? Say the input, i.e. the letters, form the word "pizza" and we have a suboptimal regular neural network. It transforms the inputs into the word "piz**s**a". It is very uncertain whether the fourth letter is "s" or "z", but decides that it is more likely an "s". It simply looks more like an "s" than a "z". Let us instead traverse the input sequentially and introduce a context as some transformation of the previous inputs. On the fourth step, the context is a transformation of "piz". Now the network can see that it is much more likely that the letter is a "z" since no word that consist of the string "piz**s**" exists in the English vocabulary. In this case, one can clearly see that traversing through the input sequentially, and at every step, can provide context as a transformation of the previous inputs that can give better results. RNNs can be used for these kinds of problems.

The math

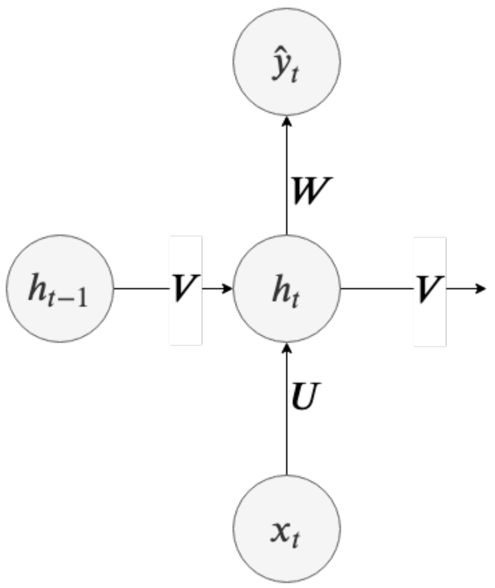


Figure 4: Visualization of the RNN at every timestep t, where the symbols in the nodes represent the output of the node and the symbols over the edges represent the parameters, i.e. the weights in the RNN

What is wanted is to transform the input $\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_T}]$ into the output $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}_{\boldsymbol{1}}, \hat{\boldsymbol{y}}_{\boldsymbol{2}}, \ldots, \hat{\boldsymbol{y}}_{\boldsymbol{T}}]$. Visualization of the RNN can be seen in Figure 4. Here, $\boldsymbol{h_{t-1}}$ denotes the "hidden state" which represents the context mentioned in section 2.1.9, i.e. a transformation of previous inputs. The output, $\hat{\boldsymbol{y}}_{\boldsymbol{t}}$, is produced at each timestep t using formulas:

$$\hat{\boldsymbol{y}}_{\boldsymbol{t}} = \boldsymbol{f}_{\hat{\boldsymbol{y}}}(\boldsymbol{h_t} \cdot \boldsymbol{W})$$

Where $\boldsymbol{f}_{\hat{\boldsymbol{y}}}$ is a differentiable, non-linear function. The sigmoid function is often used

$$\boldsymbol{h_t} = \boldsymbol{f_h}(\boldsymbol{h_{t-1}}\boldsymbol{V} + \boldsymbol{x_t}\boldsymbol{U})$$

Where $\boldsymbol{f_h}$ is a differentiable, non-linear function. The hyperbolic tangent function is often used

The newly calculated context, $\boldsymbol{h_t}$ is passed on to the next iteration in the RNN and on to the next layer which is a function of $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}_{\boldsymbol{1}}, \hat{\boldsymbol{y}}_{\boldsymbol{2}}, \ldots, \hat{\boldsymbol{y}}_{\boldsymbol{T}}]$. This means that a subset of $\hat{\boldsymbol{Y}}$ can be chosen to be further propagated through the network, meaning that the timeseries can be transformed into a timeseries of any size $\leq$ the size of the input timeseries, or into a single value.

The shortcomings of the basic RNNs

To identify the shortcomings of the basic RNN one can examine the learning process. Unrolling the graph as seen in figure 4, i.e. presents it in a way to show all of the steps of the transformation $\boldsymbol{X} \rightarrow \hat{\boldsymbol{Y}}$. The resulting unrolled network is presented in figure 5.

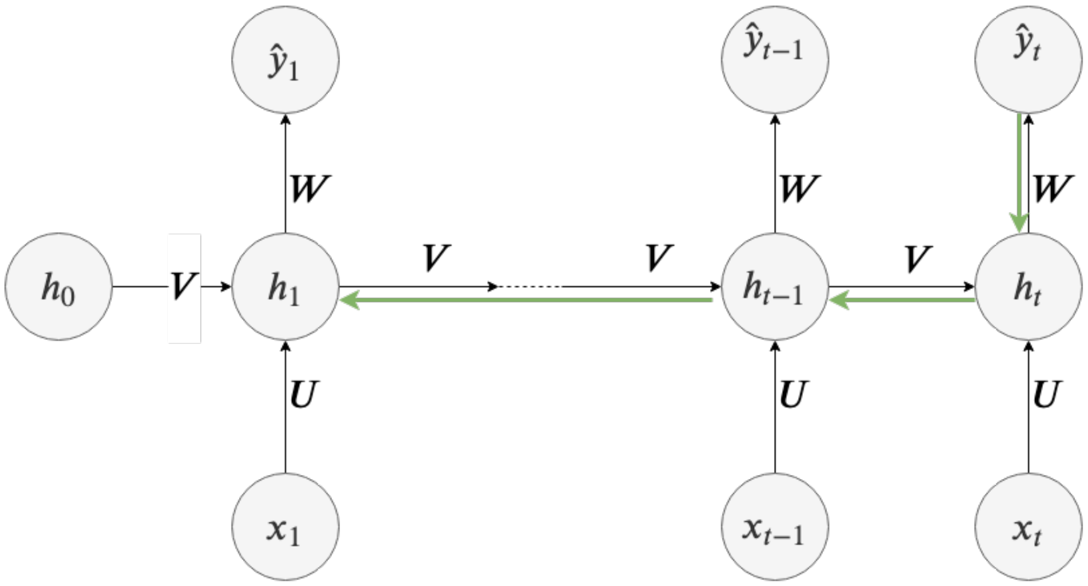


Figure 5: Unrolled RNN

Calculating the gradients of the loss with respect to the weights, $\frac{\partial L}{\partial \boldsymbol{\theta}}$ where $\boldsymbol{\theta} = \Big[\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}\Big]$. The total loss can be defined as:

$$L = \sum_{i=1}^{t} L_i$$

$$\begin{aligned}
\frac{\partial L_t}{\partial \boldsymbol{\theta}} &= \frac{\partial L_t}{\partial \hat{\boldsymbol{y}}_t}\frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{\theta}} \\
&= \frac{\partial L_t}{\partial \hat{\boldsymbol{y}}_t}\frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{\theta}} && \text{Chain rule} \\
&= \frac{\partial L_t}{\partial \hat{\boldsymbol{y}}_t}\frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_1}\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{\theta}} \\
&= \frac{\partial L_t}{\partial \hat{\boldsymbol{y}}_t}\frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{h}_t}\left[\prod_{i=1}^{t-1}\frac{\partial \boldsymbol{h}_{i+1}}{\partial \boldsymbol{h}_i}\right]\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{\theta}} && \text{Chain rule} \\
&= \frac{\partial L_t}{\partial \hat{\boldsymbol{y}}_t}\frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{h}_t}\left[\prod_{i=1}^{t-1}\frac{\partial}{\partial \boldsymbol{h}_i}\boldsymbol{f}_h(\boldsymbol{h}_{t-1}\boldsymbol{V} + \boldsymbol{x}_t U)\right]\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{\theta}} \\
&= \frac{\partial L_t}{\partial \hat{\boldsymbol{y}}_t}\frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{h}_t}\left[\prod_{i=1}^{t-1}\boldsymbol{f}_h'(\boldsymbol{h}_i\boldsymbol{V} + \boldsymbol{x}_{i+1}U)\frac{\partial(\boldsymbol{h}_i\boldsymbol{V} + \boldsymbol{x}_{i+1}\boldsymbol{U})}{\partial \boldsymbol{h}_i}\right]\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{\theta}} && \text{Chain rule} \\
&= \frac{\partial L_t}{\partial \hat{\boldsymbol{y}}_t}\frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{h}_t}\left[\prod_{i=1}^{t-1}\boldsymbol{f}_h'(\boldsymbol{h}_i\boldsymbol{V} + \boldsymbol{x}_{i+1}\boldsymbol{U})\cdot \boldsymbol{V}\right]\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{\theta}}
\end{aligned}$$

$\left|\prod_{i=1}^{t-1}\boldsymbol{f}_h'(\boldsymbol{h}_i\boldsymbol{V} + \boldsymbol{x}_{i+1}\boldsymbol{U})\cdot \boldsymbol{V}\right| \to 0$ or $\left|\prod_{i=1}^{t-1}\boldsymbol{f}_h'(\boldsymbol{h}_i\boldsymbol{V} + \boldsymbol{x}_{i+1}\boldsymbol{U})\cdot \boldsymbol{V}\right| \to \infty$ when t is large, depending on whether the values of V are large or small. This is called the vanishing gradient problem in the case where it quickly goes to 0 and exploding gradient when it quickly goes to infinity. This means that the network has troubles learning long term relationships.

In practice, basic RNNs are only able to utilize information from a few time steps back due to the problems of vanishing gradients or exploding gradients. Vanishing gradients occur due to the gradient being too small, making it difficult for the network to assess which direction the parameters should move for it to make an improvement (Goodfellow, Bengio & Courville, 2016, p. 290). In case of exploding gradients, the gradient is too large resulting in unstable learning, as the networks parameters are changed too drastically (Goodfellow, Bengio & Courville, 2016, p. 290). To mitigate the issues with vanishing and exploding gradients, variations of RNNs have been developed. One such model, a widely implemented and in some instances successful variation, is the Long Short-Term Memory (LSTM) model. Examples of the application of LSTM can be found in Section 2. Another widely implemented variation of RNN is the Gated Recurrent Unit (GRU).

2.1.10 Long Short-Term Memory (LSTM)

Long short-term memory (LSTM) model is a type of RNN model that implements the idea of self-loops to produce paths where the gradient can flow for a long duration, therefore making the model able to remember information over a long period of time and acquire knowledge across multiple time steps (Goodfellow, Bengio & Courville, 2016, p. 410-411 and Olah, 2015). LSTM models have shown to perform better on learning long-term dependencies than other comparable recurrent neural network models according to Goodfellow, Bengio & Courville (2016, p. 412). Another important feature included in LSTM models is the coping mechanism introduced to combat vanishing or exploding gradients (Goodfellow, Bengio & Courville, 2016, p 413-416), again making LSTM models suitable for capturing both long-term and short-term dependencies in the data.

The aforementioned features of the LSTM model are incorporated using states that are saved in the LSTM cells, the units of the LSTM model. To control and protect the cells and their states, three types of gates are involved: the memory/input gate, the forget gate, and the output gate (Olah, 2015 and S. S. Namin & A. S. Namin, 2018). Table 1 briefly explains the different gates.

Gate	**Description**
Memory/input gate	The gate that decides what new data to be stored in the LSTM cell.
Forget gate	The gate that decides to which degree information should be forgotten.
Output gate	The gate that decides what information should be output from the LSTM cell.

Table 1: The different gates used in LSTM.

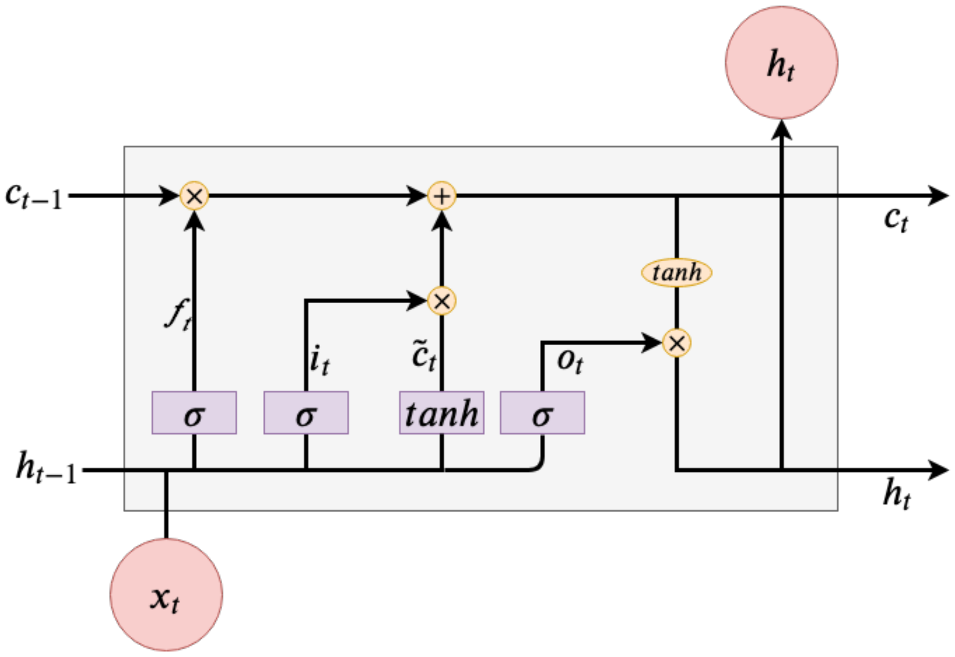


Figure 6: Illustration of an LSTM cell. Sources: Olah (2020), Chevalier (2020)

Producing the output

The mathematical procedure of producing the output is illustrated in figure 6. Producing the output $\boldsymbol{h_t}$ is done as following:

First, the forget gate vector is produced:

$$\boldsymbol{f_t} = \boldsymbol{\sigma}(\boldsymbol{W_f} \cdot [\boldsymbol{x_t}, \boldsymbol{h_{t-1}}])$$

This decides how much of each element we should retain from the previous cell state $\boldsymbol{c_{t-1}}$.

Then the adding part is calculated. The neural network produces $\boldsymbol{i_t}$ which decides which elements are needed to add information to, and a $\tanh$ neural network that produces the candidate values $\boldsymbol{\tilde{c}_t}$:

$$\begin{aligned}\boldsymbol{i_t} &= \boldsymbol{\sigma}(\boldsymbol{W_i} \cdot [\boldsymbol{x_t}, \boldsymbol{h_{t-1}}])\\ \boldsymbol{\tilde{c}_t} &= \boldsymbol{tanh}(\boldsymbol{W_{\tilde{c}}} \cdot [\boldsymbol{x_t}, \boldsymbol{h_{t-1}}])\end{aligned}$$

The vector that is added to the cell state $\boldsymbol{c_t}$ is

$$\boldsymbol{i_t} \odot \boldsymbol{\tilde{c}_t}$$

The cell state at each timestep is:

$$\boldsymbol{c_t} = (\boldsymbol{c_{t-1}} \odot \boldsymbol{f_t}) \oplus (\boldsymbol{i_t} \odot \boldsymbol{\tilde{c}_t})$$

The cell state is passed forward to the next cell and used to produce the output. For the output, the output gate is used:

$$\boldsymbol{o_t} = \boldsymbol{\sigma}(\boldsymbol{W_o} \cdot [\boldsymbol{x_t}, \boldsymbol{h_{t-1}}])$$

The output $\boldsymbol{h_t}$ is:

$$\boldsymbol{h_t} = \boldsymbol{tanh}(\boldsymbol{c_t}) \odot \boldsymbol{o_t}$$

2.1.11 Bidirectional LSTM

A bidirectional LSTM model can be viewed as an extension of the normal LSTM model. In the bidirectional model, an additional LSTM layer is added which is trained backwards on the data. Information gained from the layer training the normal way, or forwards, on the data is combined/merged with the information gained on training backwards on the data, resulting in the model potentially performing better than without this combination. This is due to the model learning one pattern of predicting when training forwards, while learning another pattern when training backwards. The combination/merging of the results from these layers can be better due to reducing the chances of overfitting on the data and reducing the variance, which is similar to how ensemble learning models can have improved performance compared to using only one instance of the model. Figure 7 depicts a bidirectional LSTM model, with the input features at time t named X_t and output at time t named Y_t. Notice that there are no direct connections between the layer that processes the data forwards and the layer that processes the data backwards. Instead, the results from these layers are combined/merged using an additional layer. Strategies for how the merging is done include summation, multiplication, averaging and concatenating the results from the layers.

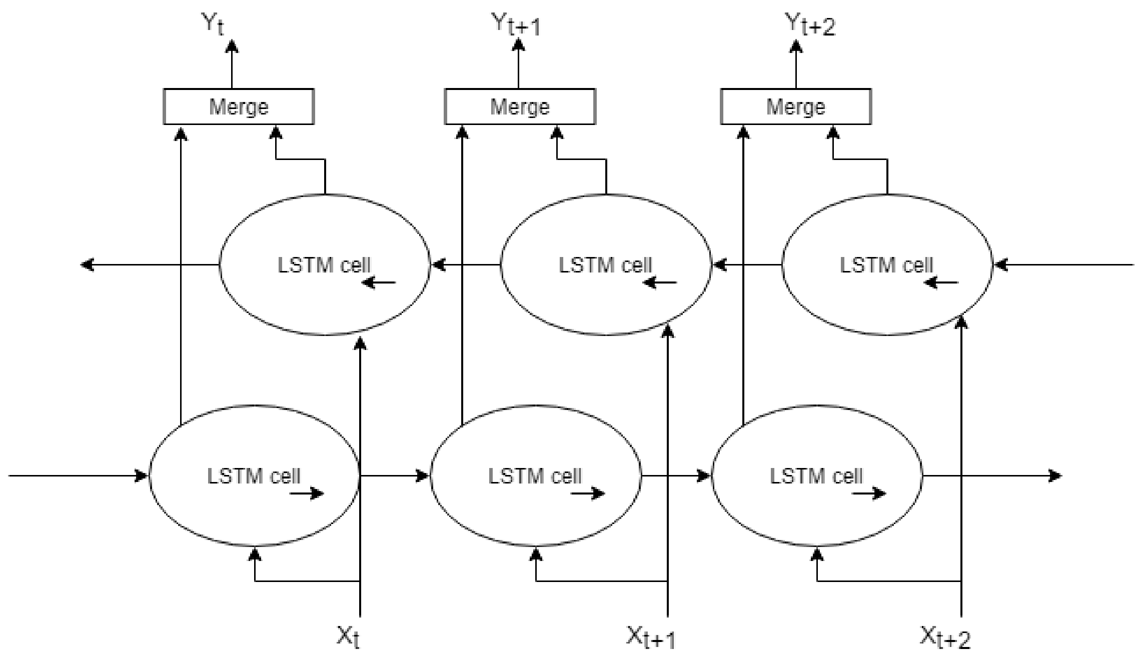


Figure 7: Illustration of a bidirectional LSTM model

2.1.12 Encoder-Decoder architecture

A model that follows the encoder-decoder architecture (also called sequence-to-sequence architecture by Goodfellow, Bengio & Courville, 2016, p. 396-398) is a model that transforms the input into another representation, which often is the final hidden state of the encoder part of the encoder-decoder model. This representation is then used by the decoder part of the model to produce the final result (Goodfellow, Bengio & Courville, 2016, p. 396-398). The representation, which can be called the context C, represents a semantic summary of the input sequence. This was originally used for language translation tasks where the input and output size of sequences could vary.

2.1.13 Adam

Adam, first introduced by Kingma & Ba (2015), is an algorithm that can be used as an optimizer in artificial neural networks, based on gradient descent which is explained in Section

2.1.8. More specifically, Adam can be viewed as an extension of stochastic gradient descent (Brownlee, 2020). In stochastic gradient descent, the data is divided into smaller subsets, also referred to as minibatches, and the algorithm performs an update on the parameters after processing each minibatch, compared to updating after processing the whole set of data.

Where Adam differs from the stochastic gradient descent is that Adam makes use of adaptive learning rates, whereas stochastic gradient descent uses a single learning rate for all updates of parameters (Brownlee, 2020). The learning rates are adapted using the first and second moments of the gradients (Kingma & Ba, 2015).

2.1.14 Autocorrelation Function and Partial Autocorrelation Function

The Autocorrelation Function (ACF) is a function that quantifies the correlation a time series contains with its lagged values. That is, given a time series $Y = (y_1, y_2, \ldots, y_t, \ldots, y_n)$, where t is an arbitrary time step, and n is the length of the time series, then ACF is defined as:

$$ACF(h) = corr(y_t, y_{t-h}), \tag{8}$$

where $corr(y_t, y_{t-h})$ describes the correlation between y_t and y_{t-h}, and h being the value representing the lag.

The Partial Autocorrelation Function (PACF) is similar to ACF, but finds the correlation with the residuals instead. That is, PACF gives the autocorrelation between a point y_t and y_{t-h} without the contribution of the points in between, meaning $(y_{t-1}, y_{t-2}, \ldots, y_{t-h+1})$.

ACF and PACF are often used to investigate how well earlier time steps can provide information on later time steps.

2.1.15 Normalization

Normalization is the act of transforming data with values that conform to different scales into data where all data conform to a common scale, without compromising the integrity of the values. One of the more common ways of normalizing is Min-Max normalization.

Min-Max normalization

Min-max normalization scales all values to be between 0 and 1. Let X be a set of values that conform to a particular scale. Min-max normalization is then achieved using the following formula:

$$\mathrm{x_{i\_normalized}} = \frac{x_i - min(X)}{max(X) - min(X)}, \tag{9}$$

where x_i is the ith value of X, $min(X)$ is the minimum value that can be found in X, and $max(X)$ is the maximum value that can be found in X.

2.1.16 Metrics

One way to evaluate machine learning models is to use metrics that measure performance. Below are the metrics that are relevant for this project.

Mean Average Percentage Error (MAPE)

Mean Average Percentage Error evaluates the performance of a model by taking the average of the absolute value of error percentage. Let $\hat{Y} = [\hat{y_1}, \hat{y_2}, \ldots, \hat{y_n}]$ be the predicted values and $Y = [y_1, y_2, \ldots, y_n]$ be the actual values, then MAPE is calculated as:

$$\mathrm{MAPE(Y, \hat{Y})} = \frac{100\%}{n} \sum_{i=1}^{n} |\frac{y_i - \hat{y_i}}{y_i}|. \tag{10}$$

Mean Average Error (MAE)

MAE evaluates the performance of a model by taking the average of the errors of the resulting predictions. Let $\hat{Y} = [\hat{y_1}, \hat{y_2}, \ldots, \hat{y_n}]$ be the predicted values and $Y = [y_1, y_2, \ldots, y_n]$ be the actual values, then MAE is calculated as:

$$\mathrm{MAE(Y, \hat{Y})} = \frac{\sum_{i=1}^{n} |y_i - \hat{y_i}|}{n}. \tag{11}$$

Mean Squared Error (MSE)

MSE evaluates the performance of a model by taking the average of the squares of the errors of the resulting predictions. Let $\hat{Y} = [\hat{y_1}, \hat{y_2}, \ldots, \hat{y_n}]$ be the predicted values and $Y = [y_1, y_2, \ldots, y_n]$ be the actual values, then MSE is calculated as:

$$\mathrm{MSE(Y, \hat{Y})} = \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{n}. \tag{12}$$

Accuracy

Accuracy evaluates the performance of a model by measuring the fraction of correctly classified instances. Let $\hat{Y} = [\hat{y_1}, \hat{y_2}, \ldots, \hat{y_n}]$ be the predicted class and $Y = [y_1, y_2, \ldots, y_n]$ be the actual class, then Accuracy is calculated as:

$$\mathrm{Accuracy(Y, \hat{Y})} = \frac{\sum_{i=1}^{n}\tilde{y_i}}{n}, \text{where } \tilde{y_i} = \begin{cases} 1, & \text{if } y_i = \hat{y_i} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

Direction Accuracy (DA)

Direction accuracy will in this project be defined as the fraction of which the predicted values in a regression task move in the same direction as the actual values. Let $\hat{Y} = [\hat{y_1}, \hat{y_2}, \ldots, \tilde{y_n}]$ be the predicted values, and $Y = [y_1, y_2, \ldots, y_n]$ be the actual values. Then let $Direction(\hat{Y})$ and $Direction(Y)$ be the representation of the direction of the values of $\hat{Y}$ and Y, respectively. The direction accuracy can then be defined as:

$$\mathrm{DirectionAccuracy(Y, \hat{Y})} = Accuracy(Direction(Y), Direction(\hat{Y})), \tag{14}$$

where $Accuracy$ is defined in Equation 13 from the section above.

2.2 Additional knowledge

Knowledge that is not fundamental with regards to the core parts of the project is provided in this section. This consists mainly of methods or concepts that appear in Chapter 3, but are not carried further into other parts.

2.2.1 Logistic regression

Logistic regression is a statistical method used in machine learning as a model for classification. It is a supervised learning model (Goodfellow, Bengio & Courville, 2016, p. 140-141) that models the probability of an instance belonging to a certain class, using a logistic function to achieve this. Example of such a logistic function is the sigmoid function, which is defined by Norvig & Russel (2016, p. 725-727) as:

$$Logistic(z) = \frac{1}{1 + e^{-z}} \tag{15}$$

2.2.2 Support Vector Machines

Support Vector Machines (SVM), introduced by Cortes & Vapnik (1995), are supervised learning models that are driven by a linear function $\mathbf{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ (Goodfellow, Bengio & Courville, 2016, p. 141-143). The linear function is used to create a maximum margin separator, meaning that it finds a decision boundary with the largest distance to the instances of the different classes (Norvig & Russel, 2016). SVM models are also able to perform classification on non-linear separable data by utilizing the kernel trick. The kernel trick enables the data to be embedded into a higher-dimensional space where it can be linearly separable (Norvig & Russel, 2016, p. 744-748), making the model able to perform non-linear classification.

2.2.3 Naive Bayes

Naive Bayes is a probabilistic machine learning model that is based on Bayes' theorem and an assumption of conditional independence, meaning that a feature is independent of all other features given that the necessary conditions are met. Bayes theorem can be defined as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{16}$$

where A and B are events (Norvig & Russel, 2016, p. 495-496).

Combining Bayes' theorem with a conditional independence assumption gives the ability to perform predictions by calculating the probability of an instance having a certain class y given that the instance has the features X. That is, calculating $P(y|X)$ (Norvig & Russel, 2016, p. 496).

2.2.4 K-Neareast Neighbors

(Disclaimer: This part is taken from a previous project report (TMA4850) that one of the authors of this thesis contributed. This part, in its entirety, was written by the author)

K-Nearest Neighbors(k-NN) is a machine learning method used for classification and regression. K-NN is classified as an instance-based learner, meaning that computations are deferred until classification or regression (Witten, Frank & Hall, 2011). It is based on learning by analogy, as the result of classification or regression of an instance are based on similar instances in its knowledge, the set of instances in the training set provided (Han, Kamber & Pei, 2011). Similarity can be measured by using a distance measure, such as the normalized Euclidean distance:

$$\text{dist}(A, B)_{\text{normalized Euclidean}} = \sqrt{\frac{\sum_{i=1}^{m}(x_i - y_i)^2}{m}}, \tag{17}$$

To perform classification or regression on the given instance, the class or value of the k most similar instances are taken into consideration. During classification, a majority vote is typically used to determine the class of the given instance. That is, give the instance the class that majority of its neighbors possess. When using regression, simple linear regression might be used to determine the value of the given instance.

2.2.5 K-means

K-means is a clustering algorithm, meaning that k-means is used to partition similar instances into groups. Similarity in this case can be that these instances are of the same class or categorization. This is done by predetermining a value k to represent the amount of clusters to divide the instances into. K centroids, representing the prototype of the clusters, are then calculated. Each instance is grouped into the cluster of the centroid the instance is closest to, where closest is usually determined by a distance measure.

2.2.6 Decision tree learning

Decision tree learning is a modelling method used in machine learning to perform predictions. A decision tree is made by determining which input feature that can be used as a means to determine the class of an instance, and builds a tree with these input features as nodes. The decision tree then performs a sequence of tests based on the tree structure, eventually arriving at a leaf node, meaning a node without child nodes, deciding the class of the instance (Norvig & Russel, 2016, p. 698).

2.2.7 Random Forest

Random forest is an ensemble learning model consisting of decision trees, meaning that the model utilizes multiple decision trees to classify instances. Random forest mitigates the issue of decision trees being susceptible to overfitting on the training data and reduces the variance by combining multiple decision trees, each one being different from the others in some way, and classify the instance by combining the classification from each of the decision trees into one classification. The idea is that the collective force of the ensemble will result in better performance compared to each individual member of the ensemble.

2.2.8 Convolutional Neural Networks (CNNs)

A convolutional neural network is a class of neural networks most often used for image processing problems including facial recognition, text extraction from images and medical image analysis. CNNs consist of alternating "convolution"- and "pooling" layers that lead to a fully connected neural network. The convolution layers identify meaningful patterns in the image while the pooling layers down-sizes the image in order to reduce the effect of noise and to reduce computations in the network. Generally the earlier layers identify simple patterns such as edges and simple shapes, while the latter layers identify more abstract patterns such as complex objects like a car, a face, etc. A successful implementation of a CNN that classifies images from the ImageNet database into 1000 different classes can be seen in Krizhevsky, Sutskever & Geoffrey E Hinton (2012).

2.2.9 Pearson correlation and Granger causality

Pearson correlation

The Pearson correlation, r_{XY}, is a measurement of the linear relationship between two variables X and Y (Yeager, 2020). The Pearson correlation is defined as:

$$r_{XY} = \frac{covariance(X, Y)}{\sqrt{variance(X)} \cdot \sqrt{variance(Y)}} \tag{18}$$

Therefore, for two discrete random variables, the Pearson correlation r_{XY} becomes

$$r_{XY} = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \tag{19}$$

where n is the sample size and $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ is the sample mean

r_{XY} is always in the interval $[-1, 1]$. A value close to -1 means that the two variables are negatively correlated and a value close to 1 means that the two variables are positively correlated. An intuitive example of positive correlation could be between the variables weight and height of people. Tall people tend to be heavier than short people. An example of negative correlation could be between the heart rate of mammals and their size. Small animals tend to have higher heart rates while bigger animals tend to have lower heart rates.

Granger causality

The Granger causality test is a test used in order to identify a causal relationship between two random variables. The test is based on prediction: X “Granger-causes” a Y if past values of X contain information not present in Y that can help in predicting Y (Granger, 1969).

2.2.10 Forecasting

Forecasting is similar to prediction, but has the main purpose of predicting multiple time steps ahead with the use of available information from current and previous time steps. Forecasting is often related to working with data that has a time aspect to it, for instance time series such as weather data or stock prices. This can in many cases be seen as more complex than regular prediction due to the higher amount of uncertainty that follows from estimating further ahead.

2.3 Tools

This section briefly describes the tool libraries that are essential to this project in order to assist in understanding as the tools are mentioned throughout this thesis. An overall description of the library as well as an overview of the relevant methods or relevant tools contained in this library are provided.

2.3.1 statsmodels

statsmodels.org (2020) provides methods and tools for conducting statistical tests and statistical data exploration, as well as estimation of various statistical models. From this library of statistical tools and methods, this project utilized the following:

- *statsmodels.graphics.tsaplots.plot_acf*: Plots the autocorrelation function.
- *statsmodels.graphics.tsaplots.plot_pacf*: Plots the partial autocorrelaction function.
- *statsmodels.tsa.seasonal.seasonal_decompose*: Seasonal decomposes into components using moving averages.

2.3.2 scikit-learn

Scikit-Learn (2020) is a library that provides tools and mehods to perform machine learning in the programming language Python, built on NumPy (a library providing matrix computation and related mathematical functions), SciPy (a library providing various numerical routines) and matplotlib (a library providing tools for visualization in Python). The tools used from this library are:

- *sklearn.preprocessing.MinMaxScaler*: Transforms data into a given range, the range 0 to 1 being the default. Used to perform min-max normalization (see Section 2.1.15).
- *sklearn.linear_model.LinearRegression*: Creates a linear regression model.
- *sklearn.linear_model.Ridge*: Creates a ridge regression model.
- *sklearn.metrics.mean_squared_error*: Computes the MSE.
- *sklearn.metrics.mean_absolute_error*: Computes the MAE.
- *sklearn.metrics.accuracy_score*: Computes the accuracy.

2.3.3 TensorFlow

TensorFlow (2020) is an open source platform for machine learning, containing a comprehensive and flexible ecosystem of tools, libraries and community resources enabling development in the field of machine learning.

The main tool from the TensorFlow library of tools utilized in this project was Keras integrated with TensorFlow.

2.3.4 Keras

Keras (2020) is an application programming interface (API) with the goal of simplifying implementation of machine learning, specifically the creation of models based on neural networks. This project utilized the following tools provided by Keras to construct the LSTM models:

- *keras.layers.Input*
- *keras.layers.LSTM*
- *keras.layers.CuDNNLSTM*
- *keras.layers.Dropout*
- *keras.layers.Dense*
- *keras.layers.Bidirectional*
- *keras.layers.Embedding*
- *keras.layers.Reshape*

Section 6.2 provides further details on how these were used.

3 Related work

This chapter aims to provide insights on related activities and achievements in the literature. The focus has been on both sentiment analysis and prediction, as the foundation for this project was to investigate how additional information such as sentiment can be combined with prediction of time series. Additionally, investigating achievements in the field of predicting/forecasting cryptocurrency values was done, as this field is comparable to stock markets. Knowledge from the field of predicting/forecasting cryptocurrency could then be transferred to the field of predicting stock market prices.

3.1 Sentiment analysis

Some of the more common approaches to sentiment analysis found in the literature include SVM and naive Bayes. Applying these methods have resulted in decent results, as can be seen in Jadav & Vaghela (2016), where the authors propose an SVM model with a more optimized kernel as their best model, and Preety & Dahiya (2015), where the authors propose a modified K-means and naive Bayes algorithm and compare against SVM.

With the growing emphasis on deep learning and the possible applications, the field of sentiment analysis has also seen an increase in utilizing deep learning methods to achieve further advancement on the task. The survey commenced by Zhang, Wang & Liu (2018) mentions various approaches using deep learning to perform sentiment analysis, many of which have obtained state-of-the-art results on the various tasks in the field of sentiment analysis. Among the applications mentioned in the survey, the models having an extended amount of usage seem to be CNN and LSTM, while a few approaches implement a combination of the two models suggesting that this area might be unexplored to some extent. Some findings in the literature imply that such a combination could improve performance on sentiment analysis, including the findings of Sosa (2017) where the author achieved improvement over both models when using a combined approach. Another example showing similar results is Alayba, Palade, England & Iqbal (2018) where the authors implemented a CNN-LSTM model that acquired an improvement over their previous tested methods in which a CNN model was among those methods.

3.2 Prediction and forecasting

Prediction has been implemented in various fields, with a large amount of these being considered successful applications. Due to the growing popularity of machine learning applications, and further development of techniques and models, there are cases where multiple approaches have been applied to one single field. Such a case can be seen in Mosavi, Ozturk & Chau (2018), which is a literature review of the usage of machine learning models in flood prediction, whereas several of the approaches have reached respectable results. In Næss (2018), Multilayered ANN and LSTM were used to predict freight rates. In Kumar (2013), the author investigates the ability of machine learning to predict soccer outcomes. The mentioned examples from the literature illustrate the wide array of applications prediction has and can be implemented in.

Forecasting has seen an extended amount of application in the literature, and has been implemented in various tasks. Such tasks include forecasting of oil prices (Jian Li, Xu, Yu & Tang, 2016), where the authors utilize sentiment analysis to forecast and conclude that sentiment analysis is a viable option for forecasting in that specific task, electric load forecasting (Bouktif, Fiaz, Ouni & Serhani, 2018), financial and economical data forecasting (S. S. Namin & A. S. Namin, 2018), and solar power forecasting (Gensler, Henze, Sick & Raabe, 2016), where all three papers implement LSTM to do forecasting and achieve decent results.

Related to the stock market, Fischer & Krauss (2018) observed the superiority of an LSTM based model over a random forest, a standard deep neural network (a neural network with multiple layers between the input layer and output layer) and a simple logistic regression model for predicting the S&P 500 from the year 1992 until 2015 using historical prices. Another interesting observation they made was that LSTM decreased in profitability over time - the market became more and more efficient in relation to the LSTM used.

3.3 Prediction of events with sentiment analysis

With the vast and continuously growing amount of sentiments and opinions available on social media platforms, such as Twitter, interest in discovering ways of utilizing this information has grown considerably. One of the fields that have experienced an extensive growth of interest is the field of prediction using sentiment analysis. Predicting results of events where the public's opinion is a major contributing factor, such as polls and elections, can be seen numerous times in the literature.

For instance, in Sharma & Moh (2016) in which the authors applied sentiment analysis of tweets using dictionary based, naive Bayes and SVM solutions and achieved results that reflected the outcome of the Indian general state elections in 2016. In El Alaoui, Gahi, Messoussi, Chaabi, Todoskoff & Kobi (2018) the authors used sentiment analysis on Twitter posts in order to predict the results of the American election from 2016 with promising results. Salunkhe & Deshmukh (2017) combined sentiment analysis to extract polarity and emotions from Twitter to investigate the performance on predicting election results in the U.S and in India. In Amador, Collignon-Delmar, Benoit & Matsuo (2017), the authors implement sentiment analysis on Twitter data using SVM models to analyze the public's opinion on Brexit, and compare the results to Internet and telephone polls, concluding that such methods could be a suitable replacement for Internet polls.

Prediction of other kinds of events, such as results of American Idol (Alon, Perrigaud & Neyrand, 2013), has also been done. Predicting and forecasting stock market prices using Twitter sentiment analysis can also be seen in the literature, for instance in Kordonis, Symeonidis & Arampatzis (2016) where the authors found correlation between tweet sentiments and stock prices and concluded the results as promising based on the findings.

All of these articles have shown promising results when it comes to predicting some future event using tweets. Most of these use methods like naive Bayes and SVM for the sentiment analysis and manually analyse the results from the sentiment analysis in order to predict the event such as a president election. If one presidential candidate has more positive tweets and less negative than the other it may lead to the conclusion that the candidate has a higher probability of winning. Although when used for a regression task, the sentiment must be processed in some way in.

3.4 Predicting stock market prices using sentiment

There are numerous works that have explored finding relations between sentiments and the stock market. The increased volume of available data has made room for constant addition in this part of the literature, with several showing promising results. The usage of Twitter posts seems to be a major focus, although extracting sentiments from other sources, such as news and online forums, is also present.

Ranco, Aleksovski, Caldarelli, Grčar & Mozetič (2015) examined the relation between Twitter sentiment and the stock market. They found a generally relatively low Pearson and Granger causality between prices and sentiment. What was interesting was that they found a significant correlation at the times where tweet volumes peeked, both at expected peaks and non-expected peaks.

Bollen, Mao & Zeng (2011) is one of the most referenced works on prediction of the stock market using sentiment analysis. They used OpinionFinder, a public tool in order to analyse the emotional content of tweets and a self organizing fuzzy neural network in order to transform the sentiment into DJIA closing values. They were able to predict the next Dow Jones Industrial Average close price with a MAPE of 1.83% and a direction accuracy of 87.6%. This is clear evidence that tweets can be helpful in order to predict stock market prices.

Jiahong Li, Bu & Wu (2017) collected sentiments from investors on different online forums as a basis for an LSTMs in order to predict the direction of the CSI300, an index reflecting the top 300 stocks traded in the Shanghai and Shenzen stock exchanges. They reached a direction accuracy of 50.71% when predicting the next close price. The vastly different results between Bollen, Mao & Zeng (2011) and Jiahong Li, Bu & Wu (2017) is an indicator that this area needs more research.

Shynkevich, Coleman, Mcginnity & Belatreche (2015) experimented with a model that in addition to trading data used news and a relevance score related to the target stock in order to predict the direction of stock price movements. The news were sourced from the news database LexisNexis which contains news articles published in major newspapers. Their solution reached a direction accuracy of up to 81.63% for the WLP (Anthem, Inc.) stock.

Kordonis, Symeonidis & Arampatzis (2016) combined sentiment analysis and machine learning models such as naive Bayes and SVM to predict prices of stocks. They reached a MAPE score of 1.668%. They did not measure direction accuracy. The MAPE result lead to the conclusion that combining machine learning and sentiments is promising. It should, however, be noted that Kordonis, Symeonidis & Arampatzis (2016) only presented the results of predicting stock prices of one single day (23rd of June 2016). Results may differ when predicting over a larger time frame.

Xiong, Nichols & Shen (2016) used a single layered LSTM on Google domestic trends data in order to predict the volatility of the S&P 500 index, reaching 24% MAPE outperforming their linear and autoregressive baseline models by at least 31%. They conclude that deep learning in the presence of strong noise is promising.

Althelaya, El-Alfy & Mohammed (2018) investigated and compared different configurations of LSTM to see how they would perform on this task and different configurations of complexity. They concluded that Bidirectional LSTM was superior to a Stacked LSTM when it comes to both long- and short term predictions of stock prices, and that their deeper neural networks outperform the more shallow.

3.5 Predicting and forecasting cryptocurrency values

Forecasting and predicting cryptocurrency values have seen an increase in research in conjunction with the rising popularity of such currency from the mid 2010s and onward. Despite the rise in popularity, forecasting in the field of cryptocurrency is still a rather unexplored territory compared to similar fields, such as forecasting of stock market prices.

Abraham, Higdon, Nelson & Ibarra (2018) proposed a multiple linear regression model that transformed Twitter sentiment and Google Trends data into predictions. They concluded that price movements related to Ethereum and Bitcoin, the two largest crypto currencies, are more accurately reflected by search volumes and tweet volumes than sentiment. They observed that sentiments were positive when prices rise and fall.

In Jain, Tripathi, DwarDwivedi & Saxena (2018), the authors combined Twitter sentiment analysis with Multiple Linear Regression to predict the value of Bitcoin and Litecoin, two well known currencies in the cryptocurrency market. In this article it was observed that Litecoin was more affected by tweet sentiments than Bitcoin. It was concluded that the price of Bitcoin is dependent on other factors like mining cost.

Galeshchuk, Vasylchyshyn & Krysovatyy (2018) combined historical data of Bitcoin value and Twitter sentiment in terms of a score to forecast the values and concluded that including sentiment in some way can have a positive effect on forecasting and prediction.

4 Data

One of the most important aspect when working with machine learning of any kind is access to useful and reliable data for the task at hand. In this chapter, the data and the data sources will be described, as well as any preprocessing steps deemed necessary to make the data useful for the task. Additionally, this chapter will cover the initial data analysis done to better understand the data.

Data values that this project aims to be able to predict will be referred to as outputs or output features, while data used to learn how to produce the output will be referred to as inputs, input features or simply features.

4.1 Data sources

A part of this project was to investigate whether data available to the public could be utilized in a way to predict prices of stocks to an adequate degree. The focus has therefore been on finding data sources that can be easily accessed by anyone, in addition to containing data that could possible improve the performance of the task.

4.1.1 Investing.com

Investing.com (2020) is, among other things, a global financial portal providing access to various information regarding the global financial markets, including analysis, news and technical data.

Investing.com was founded in 2007, and according to their usage data, has an estimate of 20 million unique visitors and 1 billion page views per month.

4.1.2 StockFluence

StockFluence (2020) is a tool providing sentiment analysis on stocks from some of the biggest companies on the global scale. StockFluence claim to be able to predict stock price movement with an accuracy of 70%. This is achieved by analysing over 400 thousand articles and 1 million tweets daily. In total, almost 1 billion articles have been analyzed, in addition to over 2 billion tweets.

4.1.3 Google Trends

Google Trends (2020) provide access to relative scores of search terms on Google Search Engine. The scores reflect the relative popularity of the search term. As Google has been providing the most popular search engine for more than a decade, using Google Trends to investigate the popularity of a search term seems reliable to a sufficient degree.

4.2 Description of the data

Data related to 15 different stocks were gathered for this project. The stocks are represented in a variety of different market categories, including, but not limited to, technology, retail, pharmacy, and e-commerce. Table 56 in the Appendix presents all the stocks included in the project along with a small description of the company the stock belongs to.

The following sections will present the different types of data gathered for each stock.

4.2.1 Historical trading data

Historical stock data were obtained from Investing.com. The historical data contain information related to the stock from every day available. The data considered utilized and experimented upon are presented and shortly described in Table 2.

Name	**Description**
Open	The price of which the stock is first traded at on the opening of the corresponding trading day.
High	The highest price the stock is traded at for the corresponding day.
Low	The lowest price the stock is traded at for the corresponding day.
Price	The final price of which the stock is traded at for the corresponding day.
Volume	The amount of stocks traded on the corresponding day.
Change %	The change in price between the previous and the corresponding day, given in percentage.

Table 2: Data from Investing.com used in this project

The “change %” feature was used to generate an additional feature to reflect the direction instead of the change of price given in percentage. The direction was specified to have the value 1 if the change was equal to or higher than 0, and would be 0 otherwise.

In the additional experiments, a feature called “change” was added, reflecting the actual price change between between the price of the day before and the current price. Later, additional features representing change was added as well, including “open_close_change” depicting the difference between “open” and “price” of the current day, “high_close_change” depicting the difference between “high” and “price” of the current day, “low_close_change” depicting the difference between “low” and “price” of the current day, and “trendscore_change” depicting the difference between the value of “trendscore” of yesterday and the current “trendscore”.

4.2.2 Stock sentiment data

The stock sentiment data were provided by Stockfluence. The data consist of information Stockfluence is willing to provide to the public. Data utilized from the stock sentiment data are presented and shortly described in Table 3.

Name	**Description**
Positive	The amount of articles and Twitter posts related to the stock that were categorized as positive by Stockfluence’s sentiment analyzer.
Neutral	The amount of articles and Twitter posts related to the stock that were categorized as neutral by Stockfluence’s sentiment analyzer.
Negative	The amount of articles and Twitter posts related to the stock that were categorized as negative by Stockfluence’s sentiment analyzer.

Table 3: Data from Stockfluence used in this project

To have an alternative way of using the stock sentiment data, the values of “positive”, “neutral” and “negative” were used to represent the proportion of positive, neutral and negative posts. For instance, if the amounts were “positive”=21, “neutral”=61 and “negative”=2, the values would be changed to “positive_prop”=25.0%=0.25, “neutral_prop”=72.6%=0.726

and “negative_prop”=2.4%=0.024. These alternative data were used in addition to the original data, which gave not only more features, but also the possibility of different stock sentiment data combinations.

Other data provided by Stockfluence included a score of the sentiments, a number representing the change in sentiment from day to day, score of the strength, reach and passion of the sentiments, among other information. These data were excluded as they were not provided consistently over the timeline to be analyzed, or because they were deemed unnecessary for the experiments.

4.2.3 Trend data

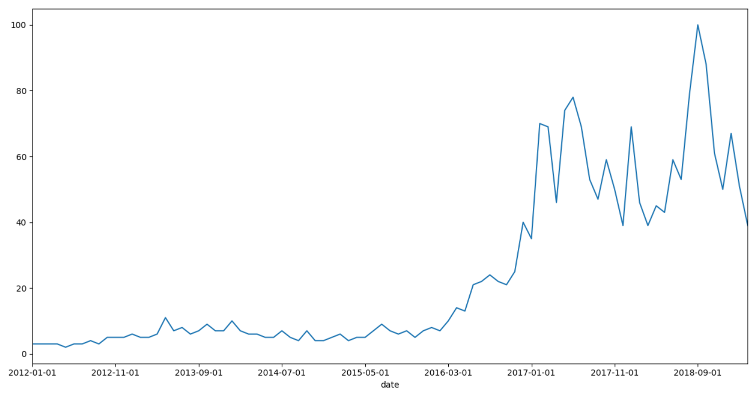
The trend data was provided by Google Trends, and were used to see if search popularity or frequency could have correlation with stock prices.

Google Trends provide a score of each time point of the search term at interest, relative to the highest and lowest scores of the time period. Because of limitations on the functionality provided by Google Trends, mainly due to the restrictions on the timeline of daily scores, some preprocessing were necessary to get a better approximation of the scores for the timeline to be examined in this project. To obtain these approximations, the data were adjusted according to the method suggested by Bewerunge (2018). Google Trends aggregates the days into larger groups, for instance into weeks or months, when larger timelines are examined. To circumvent these limitations, data from each month during the whole time period were downloaded separately to get the daily relative scores, then the monthly scores for each month during the whole timeline were downloaded together to get the monthly scores relative to each other. The daily data were then scaled according to the monthly scores of their associated month, using the formula:

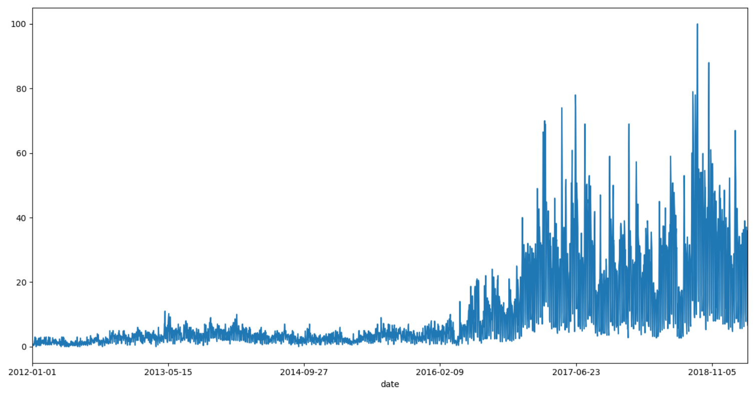
$$\mathrm{DailyScore}_{\mathrm{Adjusted}}(\mathrm{d}) = \frac{\mathrm{DailyScore}_{\mathrm{Original}}(\mathrm{d}) \cdot \mathrm{MonthlyScore}(\mathrm{d})}{100}, \tag{20}$$

where d is the day of interest, $DailyScore_{Original}$ is the non-adjusted relative daily score of d, and $MontlyScore$ is the monthly score of the associated month of d.

Summarized, the data were adjusted by using the aggregated score for each month in the timeline at interest to give a better approximation of the daily data of each month in the timeline. Figure 8 is an example of the data before and after the applying the process.



(a) Trend data before: Displaying aggregated scores for each month



(b) Trend data after: Displaying approximate daily scores

Figure 8: Example of trend data before and after applying adjusting process.

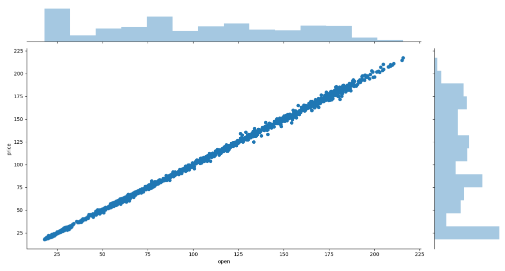
4.3 Initial data analysis

To get a better understanding of the data to be included in the experimentation phase, data analysis was performed prior to the experiments. Results from the analysis were used as guidance on data inclusion and handling, in addition to increase knowledge before and after the results.

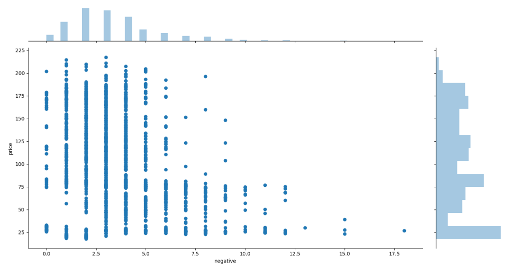
Correlation

Correlation plots were used to inspect the linear relationships between the input features and output. Additionally, the plots were used to determine the input features which clearly did not have linear relationships with the output, with the goal to experiment with these to see whether a non-linear relationship could be captured by the model that could possibly improve the results.

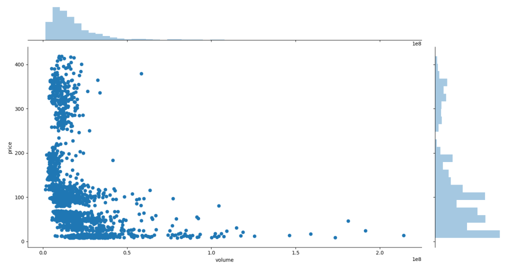
The analysis resulted in evidence that some of the input features had strong linear relationship with the outputs. Figure 9a shows an example of a clear linear relationship that is present in the data set. Multiple input features exhibit similar behavior with the output "price", including "low" and "high". Other input features seem to show that no linear correlation is present between them and the output "price", including "volume" and "negative". Examples of such results are shown in figure 9b and 9c.



(a) Plot displaying linear relationship is present between values for "open" and "price".



(b) Plot displaying no linear relationship is present between values for "negative" and "price".



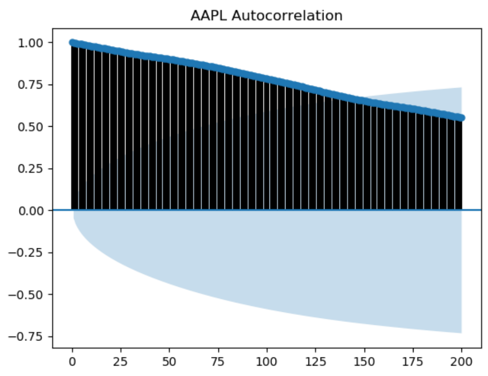
(c) Plot displaying no linear relationship is present between values for "volume" and "price".

Figure 9: Examples of plots showing that linear relationship is present and not present.

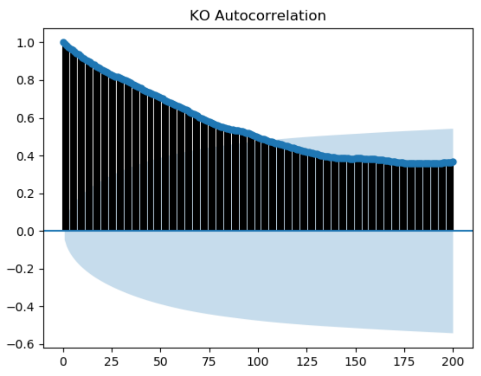
Autocorrelation

When working with time series data, adding output from previous time steps, also known as lags, as additional input features might be beneficial in some cases. This adds information and patterns the prediction model can make use of to improve the output prediction. To better the understanding of the data and see if there were any obvious indications that including lags would be beneficial, Autocorrelation Function (ACF) plots and Partial Autocorrelation Function (PACF) plots were produced using functionality provided by statsmodels.org (2020).

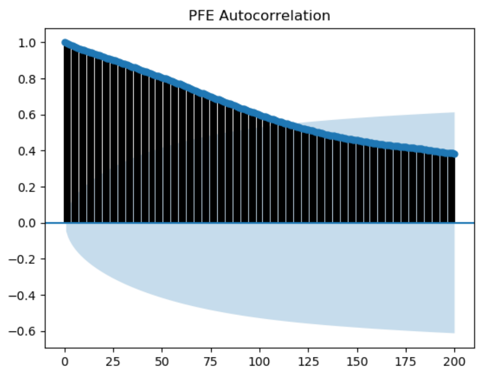
The results from the ACF plots of the different stocks suggest that most stocks behave in a similar manner when it comes to autocorrelation. That is, the correlation seems to drop slowly as the lags increase, and there are no noticeable drop or increase in autocorrelation value. This is illustrated in Figure 10, which depicts the general trend across all stocks with regards to ACF.



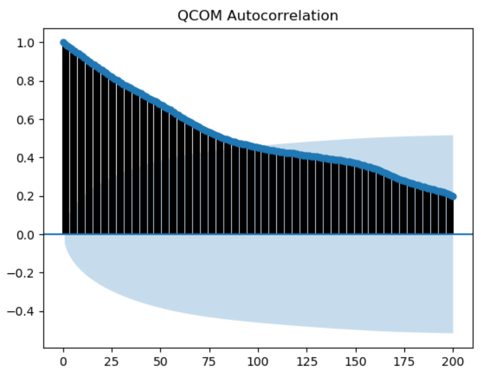
(a) AAPL



(b) KO



(c) PFE



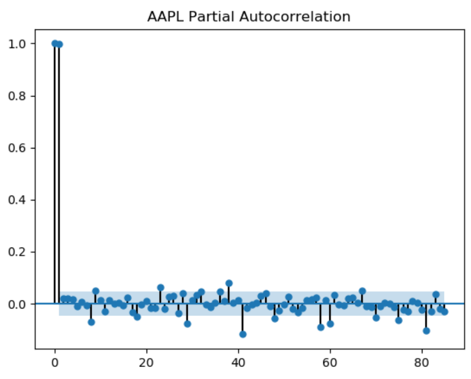
(d) QCOM

Figure 10: ACF plots of prices of the stocks AAPL, KO, PFE and QCOM

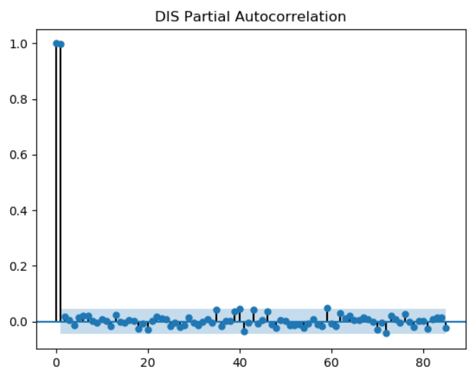
The kind of behavior described above and illustrated in Figure 10 suggests that the time series data used in this project are non-stationary, which is important to keep in mind. The results suggest that if lagged variables are to implemented, only including a few time steps backwards might improve the performance.

To further analyze and understand the data, PACF was applied. The results from the PACF plots suggested that only one time steps backwards could be useful for prediction. Figure 11 shows some of the results. Results from the PACF plots suggest that most stocks behave in a similar way when it comes to partial correlation. In a few cases, as can be seen in Figure 11c, there seem to be some movement when the lags increase, which can be interesting to investigate further. However, the other results from Figure 11 seem to be the norm.

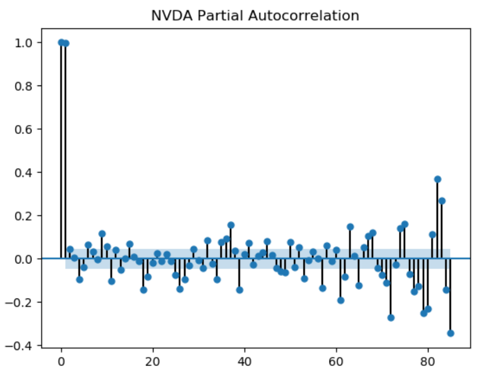
It is important to note that the results from the ACF and PACF analysis do not determine that there are no useful information in the earlier time steps, only that the data have the strongest correlation with the directly preceding time step. Including earlier time steps might still lead to improved performance. Due to the lack of evidence in the amount of lagged variables that could prove useful, this project will include a maximum of two earlier time steps for investigative purposes.



(a) AAPL



(b) DIS



(c) NVDA

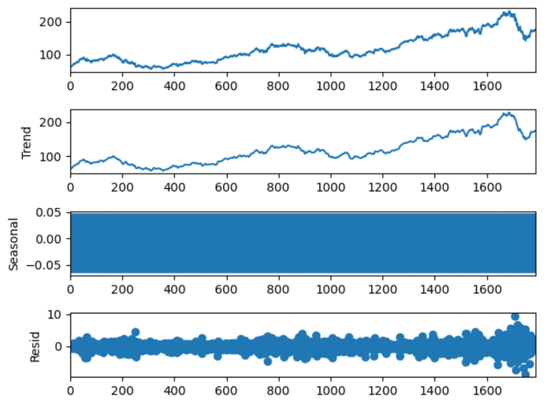
Figure 11: PACF plots of prices of different stocks

Seasonal decompose

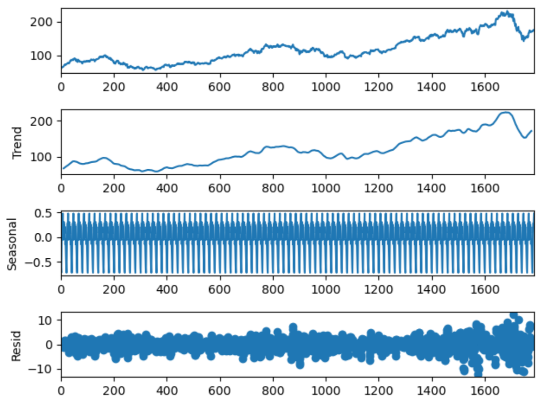
Seasonal decomposition is the act of decomposing data into different components that are meant to reflect different patterns present in the data. This project utilizes this method to assist in understanding the nature of the data better. The analysis was done using the seasonal_decompose tool provided by statsmodels.org (2020), which calculates the decomposition using moving averages. The observed values were decomposed into:

- **Trend:** The trend shows the increasing and decreasing direction of the data
- **Seasonal:** The component reflecting the seasonal variations of the data
- **Resid:** The random variations of the data, also known as noise

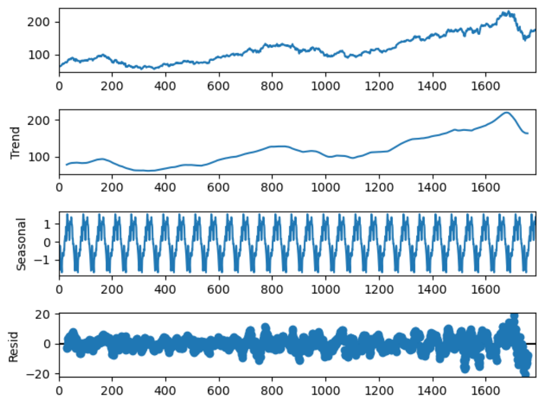
Four different values for period were chosen for analysis. These are 5, 20, 60 and 250, and correspond to the amount of trading days in a week, month, quarter and year, respectively. None of the values are exact values, as the exact number of trading days in a certain period depends on a number of different factors, such as holidays, leap years and so on. They are, however, close to the actual numbers and can assist in understanding the data that are being dealt with. An example from the results is presented in Figure 12, illustrating how the different periods affect the seasonal decompose results.



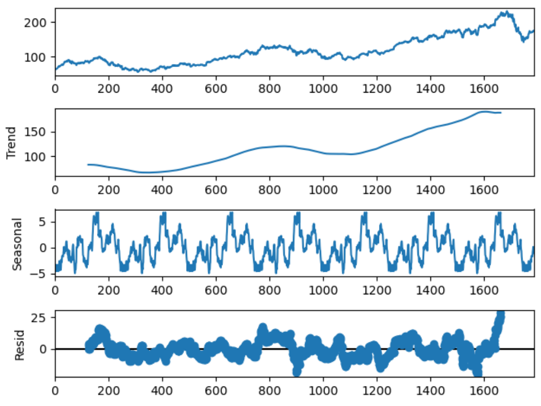
(a) With period 5.



(b) With period 20.



(c) With period 60.



(d) With period 250.

Figure 12: Seasonal decompose on the stock AAPL with different periods.

What can be seen from this part of the analysis is that a large part of the stocks have a general upwards trend. There are stocks that fluctuate, and even stocks that have a downwards trend. This indicates that in terms of trend, there might not be a general description that will hold true for all stocks. The seasonal components seem to be relatively low for all stocks compared to the trend and observed max values, indicating that stock prices are little affected by seasonal variations. This seems to hold true for all values of period analyzed. These results indicate that overall, stock prices do not follow a overarching seasonal pattern. Furthermore, the various resid results suggest that there is a prevalent presence of random variations in the data, suggesting the difficulty of working with stocks and illustrate the uncertainty factor related to this task.

4.4 Data preprocessing

Some preprocessing were necessary in order to use the different models selected for this project. The applied processes will be briefly described below. Reasoning behind these decisions will also be presented.

Normalization

In order to make progress with several models, and improve performance, it was necessary to normalize the data. Normalization of data is the act of adjusting the data to conform to a chosen scale (see Section 2.1.15). This can assist in obtaining improved result and an increased speed of learning (Stöttner, 2020).

Leaving the data as is can have a negative impact on performance that can be introduced due to different features operating on different numerical scales. For instance, the "volume" feature have values in the millions, compared to "change %" that only has values between 0 and 1. The difference in magnitude can cause models to incorrectly assume that "volume" is a more important feature in terms of prediction, due to the difference in magnitude. This can be because of values that operate on a larger magnitude also change with greater magnitude. These changes can cause the models to regard these features with higher importance than what actually is the case. Normalizing the features to a common scale can assist in mitigating this problem.

The other positive outcome when normalizing the data is that it can help models that have weights of some kind, such as the LSTM models that will be used in this project, find adequate weights in less time. Normalizing the data to only have values between 0 and 1, for instance, allows for the usage of higher learning rates and makes weights easier to initialize.

Also, some methods, such as ridge regression, require normalization to perform properly. This is due to the regularization in ridge regression, which may lead to different and unfair regularization when features conform to different scales.

In this project, all data have been normalized using min-max normalization (Section 2.1.15). Further details on implementation can be found in Section 6.1.1.

5 Architecture

The system in this thesis is a neural network and in this chapter the structure and the rationale behind the decisions will be presented. The objective of the system is to transform time series of stock signals into accurate predictions of stock prices. The system implemented can be divided into two independent modules: the context module (described in Section 5.1) and the prediction module (described in Section 5.2). In this thesis the two different subsystems - one with the context module, depicted in Figure 14, and one without, depicted in Figure 15 - will be analyzed and compared.

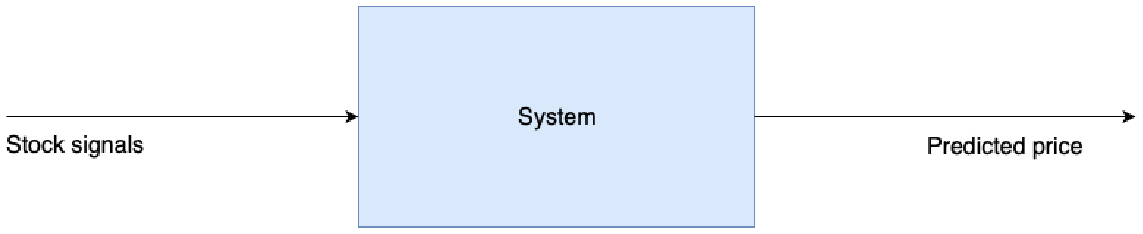


Figure 13: System with input and output

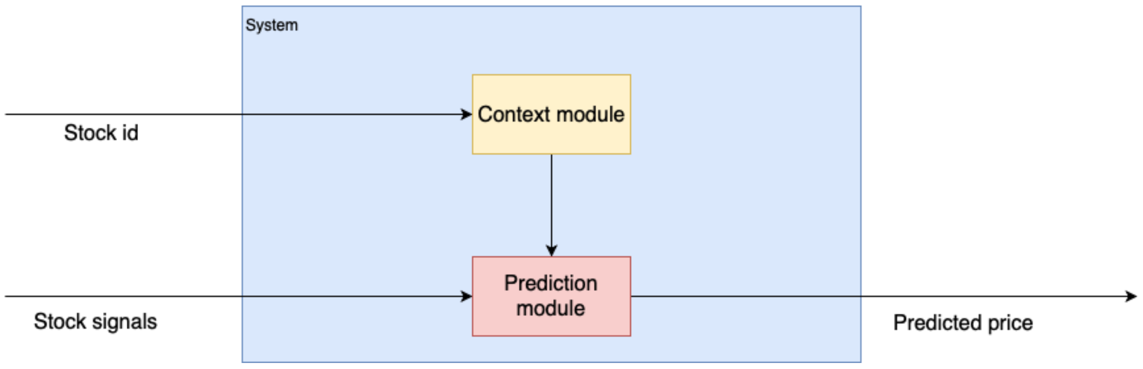


Figure 14: Subsystem 1: With the context module

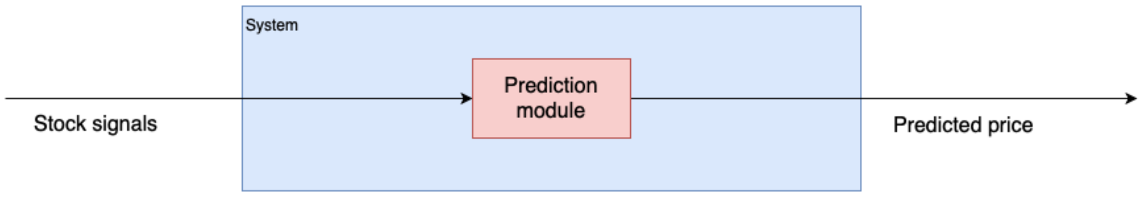


Figure 15: Subsystem 2: Without the context module

5.1 Context module

To possibly assist the LSTM models in making better predictions, context information will be included. This context might help the models understand that each stock should be treated differently.

Additionally, are there some similarities between how different stocks behave? It seems reasonable to believe that there exists a stock A that behave more similarly to some stock B than to some stock C in terms of price movements. Maybe stocks A and B are related to similar companies in the same section of the market, while stocks A and C are related to wildly different companies. Let us posit a hypothesis: "Some stocks behave more similar to each other than they do other stocks". If this hypothesis is correct, can such a relationship be captured, and can one transfer knowledge of one stock and apply it to another stock?

Inspired by the encoder-decoder architecture, this thesis will propose a context module that will process the name of the stock semantically, in the form of an embedding. This embedding is a semantic vector in the form that vectors that are close to each other are more similar than vectors that are far from each other. Instead of the context module processing the whole sequence of features like in traditional encoder-decoder architectures it will only process the name of the stock to make a representation which will be used as the initial state for the LSTM based network. The ideas is that this will enable the LSTM based network to tweak the behavior depending on what stock that is currently being predicted, since each stock name will produce different initial states. Additionally, this might enable the embedding to produce representations that allow for the LSTM based network to make use of the hypothesis described in the previous paragraph. Description on how the context module was combined with the LSTM based networks is provided in Section 6.2.9, while Section 7.2.1 presents how the context module fits in the experimental plan.

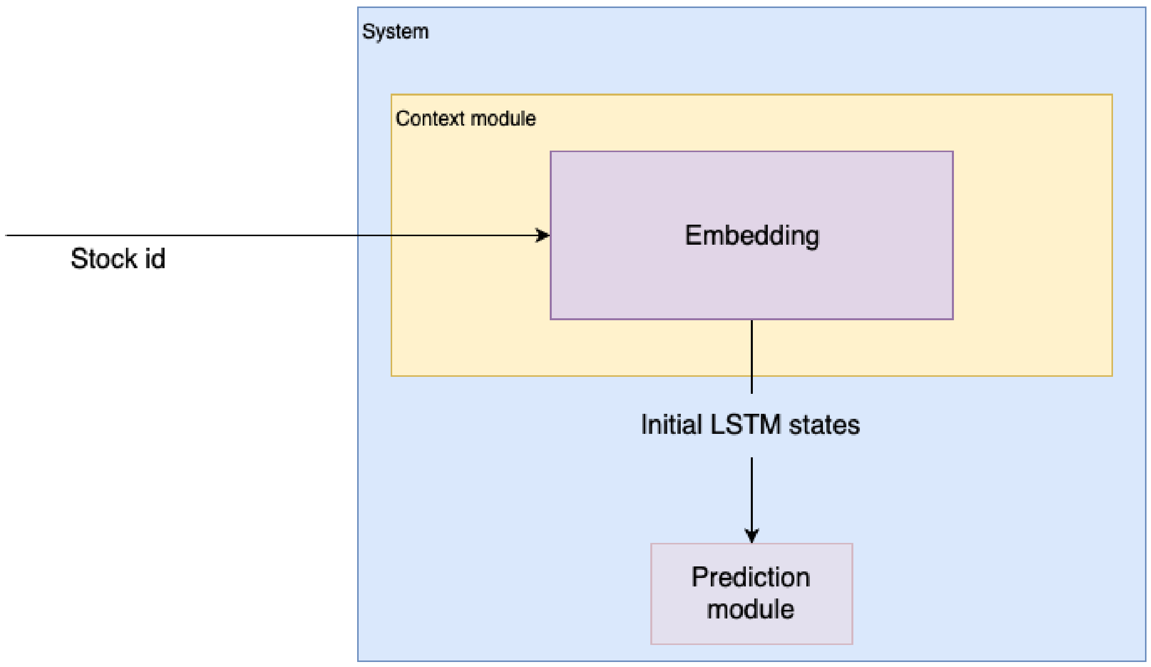


Figure 16: The context module

5.2 Prediction module

The prediction module is the crux of the system and will predict future prices by processing stock signals, sentiment and search trend. Two different implementations of the prediction module will be compared and analyzed, one with regular LSTMs and one with bidirectional LSTMs. The vanilla LSTM can be seen as a stacked LSTM with one layer. The initial state of the LSTMs is either the zero vector if it is not provided with any context or the output of the context module if provided. Different numbers of LSTM/bidirectional LSTM layers will be tested, in addition to different configurations of dropout layers.

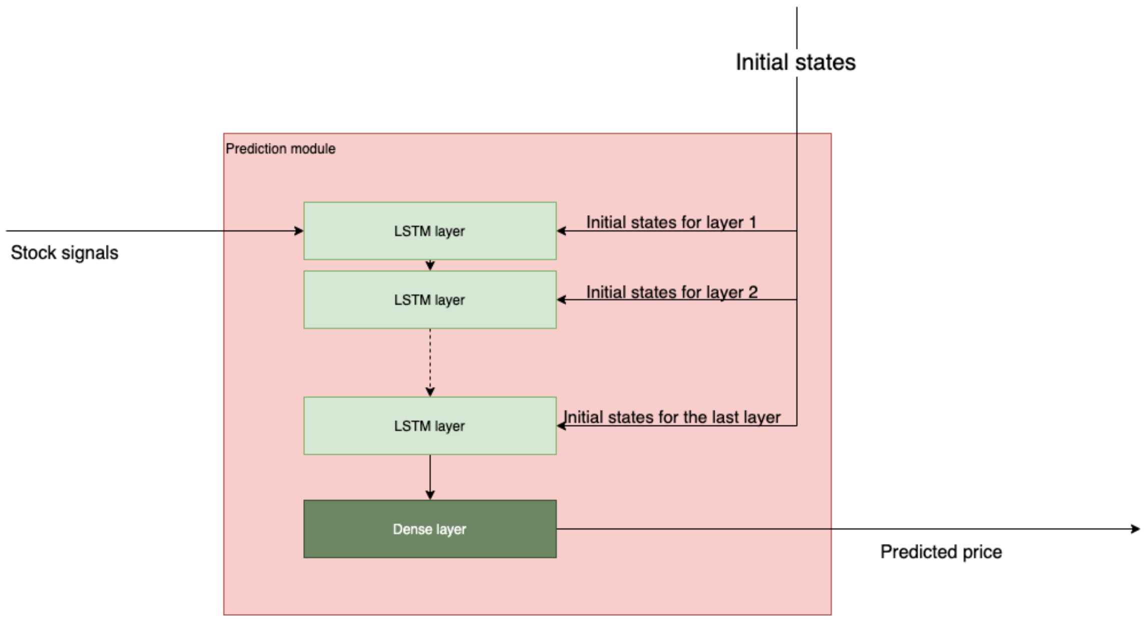


Figure 17: Prediction module with stacked LSTMs

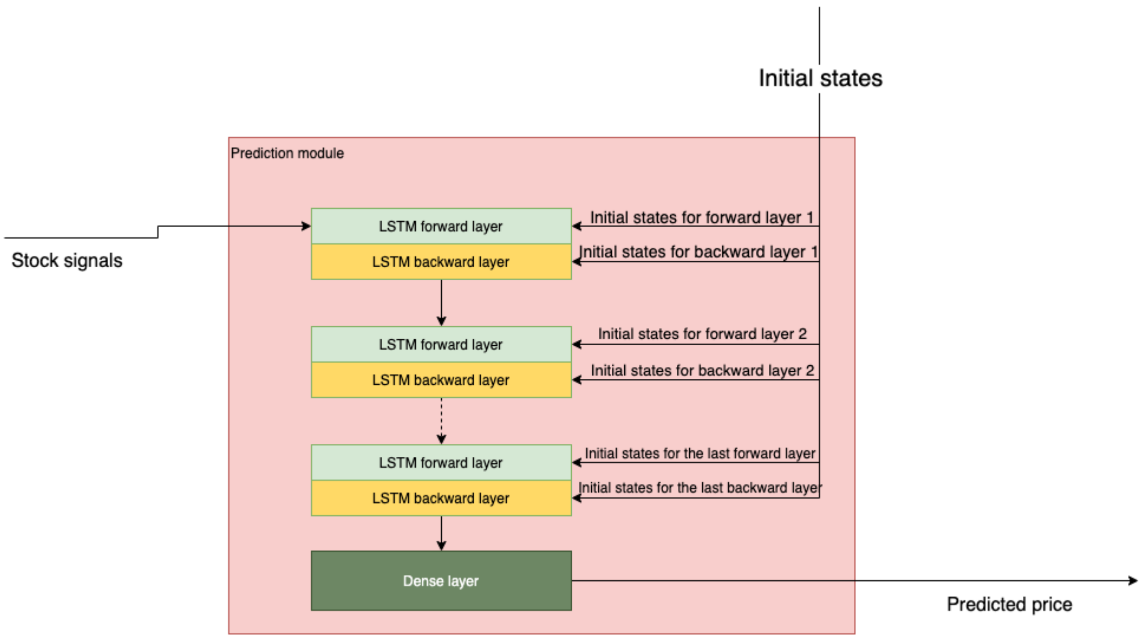

Initial states
Prediction module
Stock signals
LSTM forward layer
LSTM backward layer
Initial states for forward layer 1
Initial states for backward layer 1
LSTM forward layer
LSTM backward layer
Initial states for forward layer 2
Initial states for backward layer 2
LSTM forward layer
LSTM backward layer
Initial states for the last forward layer
Initial states for the last backward layer
Dense layer
Predicted price

Figure 18: Prediction module with stacked bidirectional LSTMs

5.3 Multiple outputs

In an attempt to increase the performance on direction accuracy, systems trained to predict both the next day prices and the direction were implemented. The overall system remained the same, with only an additional value that the system outputs, as illustrated in Figure 19.

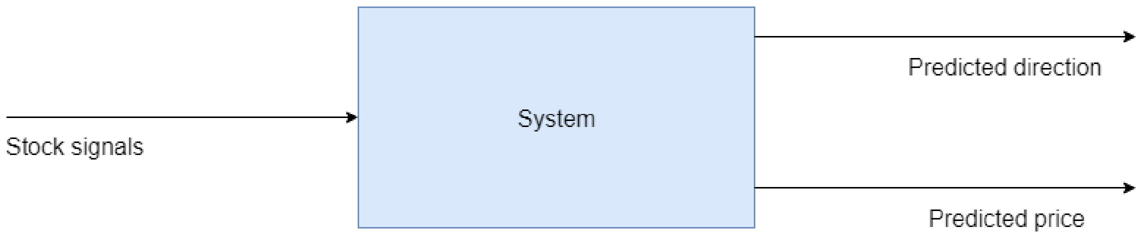


Figure 19: System with input and two outputs

6 Experimental setup

This chapter introduces the setup and implementations necessary to conduct the experiments. The chapter is separated into three parts, the first part presents the setup associated with the data, the second part presents how the models are implemented in order to produce the results, while the last part will explain the metrics utilized in evaluation, the tools used and their implementation.

6.1 Data related setup

6.1.1 Normalization

In order to normalize the data, Scikit-Learn's *sklearn.preprocessing.MinMaxScaler* with default parameters was applied.

$$\begin{aligned} \text{Let } n_s &= \text{number of stocks} \\ n_{train} &= \text{number of training time steps} \\ n_{test} &= \text{number of validation/test time steps} \\ n_f &= \text{number of features} \end{aligned}$$

The full data set is represented by a numpy array of shape (n_s, $n_{train} + n_{test}$, n_f). In order to scale the data set, the full data set is divided into n_s arrays of shape (1, $n_{train} + n_{test}$, n_f) in order to scale each stock individually. Only the training set will be used for fitting the scaler in order to prevent peeking: For each of these arrays of size (1, $n_{train} + n_{test}$, n_f), we use the first (1, n_{train}, n_f) to fit the *sklearn.preprocessing.MinMaxScaler*. This is done to prevent the models from acquiring information that would not be present otherwise, such as the mean of the whole set instead of just the training set. The whole array is then transformed using the fitted scaler. Each feature is fitted and transformed separately; the fitting/transformation of a feature is independent of another feature. E.g. the resulting scaled price is dependent only on the price in other time steps, and not on other features such as volume. Normalization was applied to both the input features and the output.

6.1.2 Dividing data into training, validation and test sets

The dataset $\{X, y\}$ was at first divided into two sets. The first set was used to train the model, hereby called the training set or training data $\{X_{train}, y_{train}\}$. The second set was held out until performance evaluation time and used to evaluate how well the model generalises, i.e. how accurately it predicts on data it has never seen before. This data was called the test set $\{X_{test}, y_{test}\}$. The dataset was then defined as $[\{X_{train}, y_{train}\}, \{X_{test}, y_{test}\}]$. The general, incremental method to creating a good model is to train the model on the training set, then validate it using a validation set. Then modify some of the hyperparameters to see whether one can achieve better validation results. Repeat this process until the model is satisfying. If this method was to be used with the training set and test set only, it would introduce some bias in that the hyperparameters are updated in order to increase the accuracy on the test set - the model has been tweaked to perform as good as possible on the test set, deeming the results biased and skewed, ultimately not representative of real results. Therefore, the training set was further divided into a training set and a validation set, resulting in the dataset being split into three parts - a training set, a validation set and a test set, $[\{X_{train}, y_{train}\}, \{X_{val}, y_{val}\}, \{X_{test}, y_{test}\}]$, where $\{X_{train}, y_{train}\}$ precedes $\{X_{val}, y_{val}\}$ in terms of time, and $\{X_{val}, y_{val}\}$ precedes $\{X_{test}, y_{test}\}$ in terms of time. Figure 20 depicts the described scenario.

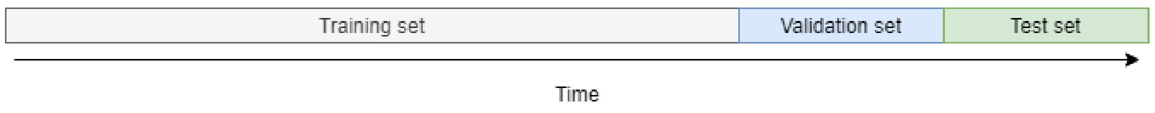


Figure 20: The complete dataset divided into a training set, a validation set and a test set

The test set is untouched when optimising the model by hyperparameter tuning or feature searching, and can therefore be used on the final model to evaluate the model in an unbiased way. The final results on the test set will be used to compare the models in this thesis as well as to compare results to related work whenever possible. The ratio {80%, 10%, 10%} will be used for the training set, validation set and the test set respectively when tuning hyperparameters, analyzing features and training and evaluating the LSTM models (training set for training, validation set to decide early stopping and for experimentation, test set for evaluation), while the ratio 90%, 10% will be used during evaluation of the baseline models. As each stock has a total of 1660 data points, the described way of dividing the data set will result in a training set of 1328 data points and a validation set of 166 data points and a test set of 166 data points. The ratio 90%, 10% will result in a training set at 1494 data points, with the test set at 166 data points.

6.2 Model implementation

A detailed description on how the different models have been implemented is presented in this section. This includes the off-the-shelf methods, tools and packages utilized, in addition to any important parameters provided to the aforementioned methods, tools and packages. These tools and packages consist primarily of Keras, NumPy (2020) and Scikit-Learn (2020). Parameters were kept as default, unless specified otherwise.

All LSTM configurations were trained the same way. This was done by training on the whole set of training data, without dividing the data according to stocks and training on each stock individually. All results using the LSTM models were produced in a similar manner. Producing results with the baseline models differed slightly from the approach with the LSTM models, which is explained in their respective sections below.

6.2.1 Simple 1-step-behind model (naive model)

This is a simple model that outputs the current price, which is given as an input feature, as the predicted price of the next day.

6.2.2 Linear regression

The linear regression model was implemented using *sklearn.linear_model.LinearRegression*, the linear regression model provided by Scikit-Learn.

Each stock had a linear regression model with default parameters and the model was built with the training data of the corresponding stock using the function *fit*. The results were then produced using the function *predict* on the corresponding test data.

All features available, specified in Section 7.3.3, were used as input to the linear regression model.

6.2.3 Ridge regression

The ridge regression model was implemented using *sklearn.linear_model.Ridge*, the ridge regression model provided by Scikit-Learn.

A parameter called *alpha* is included in the provided model that represents the strength of the regularization, which will be depicted as α in this project. The α value was initially kept as default ($\alpha = 1.0$), to see how the model would perform. Section 8.1.5 will describe experimenting with the value for α further.

For each stock, a ridge regression model was created and trained upon the corresponding training data with the provided function *fit*. All models utilized the same value for α. Generating the results for a specific stock were done by running *predict* with the associated ridge regression model on corresponding test data.

The ridge regression model utilized all features as input features. These are specified in Section 7.3.3.

6.2.4 Random guessing

The random guessing model takes in the current price as a feature, and predicts the next day price by either adding or subtracting from the current price, with both cases having an equal probability of happening.

6.2.5 Vanilla LSTM

The vanilla LSTM model was implemented using the following layers, all provided by Keras:

- Input layer: *keras.layers.Input*.
- LSTM layer: *keras.layers.LSTM* or *keras.layers.CuDNNLSTM* depending on the hardware used, with *return_sequences=True* and *return_state=True*.
- Dropout layer: *keras.layers.Dropout*.
- Output layer: *keras.layers.Dense*, with *activation="linear"*.

Figure 21 illustrates how the layers mentioned above are connected in the implementation.

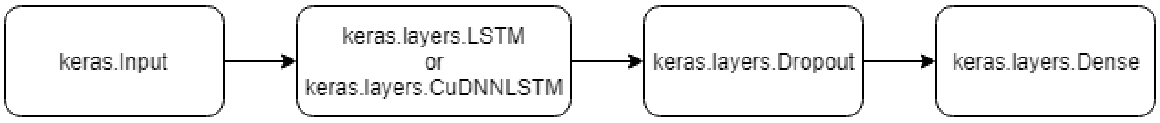


Figure 21: Connections between layers in the vanilla LSTM model

The model input was specified to take in the input features and states, while the model output was specified to be the output from the output layer and the newly computed states returned by the LSTM layer. The initial states were set to either all zeros or to the initial state outputted from the context module, depending on the configuration.

Vanilla LSTM with 2 output layers

The vanilla LSTM model with 2 output layers was implemented similar to the vanilla LSTM model, except that it had an additional output layer using *keras.layers.Dense* with *activation=“sigmoid”*. Figure 22 illustrates the change applied to the vanilla LSTM model.

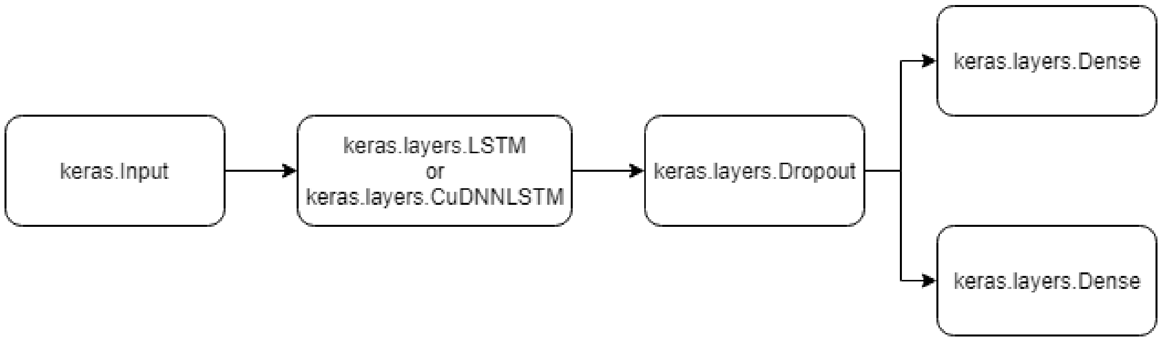


Figure 22: Connections between layers in the vanilla LSTM model with two output layers

6.2.6 Stacked LSTM

The stacked LSTM model was implemented using the following layers, all provided by Keras:

- Input layer: *keras.layers.Input*.
- LSTM layers: *keras.layers.LSTM* or *keras.layers.CuDNNLSTM* depending on the hardware used, with *return_sequences=True* and *return_state=True*.
- Dropout layer: *keras.layers.Dropout*.
- Output layer: *keras.layers.Dense*, with *activation=“linear”*.

Figure 23 illustrates how the layers mentioned above are connected in the implementation.

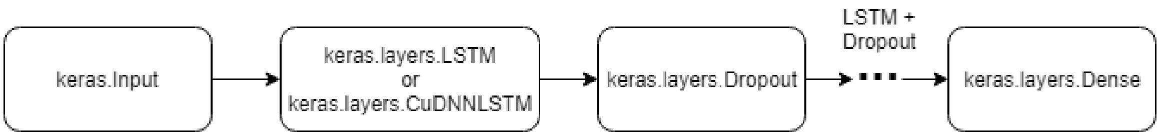


Figure 23: Connections between layers in the stacked LSTM model. The dots represent the repeat of LSTM and Dropout layers.

The model input was specified to take in the input features and states, while the model output was specified to be the output from the output layer and the newly computed states returned by all the LSTM layers.

6.2.7 Bidirectional LSTM

The bidirectional LSTM model was implemented using the following layers, all provided by Keras:

- Input layer: *keras.layers.Input*.
- Bidirectional wrapper: *keras.layers.Bidirectional*, with *merge_mode="ave"*, meaning averaging.
- LSTM layers: *keras.layers.LSTM* or *keras.layers.CuDNNLSTM* depending on the hardware used, with *return_sequences=True* and *return_state=True*.
- Dropout layer: *keras.layers.Dropout*.
- Output layer: *keras.layers.Dense*, with *activation="linear"*.

Figure 24 illustrates how the layers mentioned above are connected in the implementation.

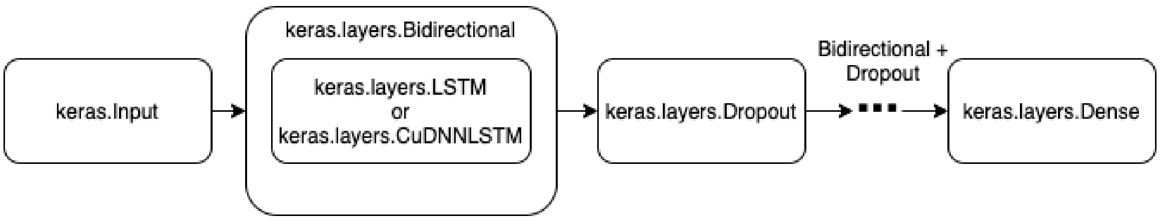


Figure 24: Connections between layers in the bidirectional LSTM model.

Bidirectional LSTM prediction

Prediction was carried out in another way for the bidirectional models than the other models because of the backwards LSTM layer. Because of this backwards layer, it was important to not feed the whole data set at once during prediction. If we are predicting the next price at time t, then the model will use features from the future if possible because of the backwards LSTM layer. Therefore in order to predict a time series of stock prices, we have to feed the time series incrementally, starting with one time step, then two time steps, then three time steps, etc. and using only the last predicted price each increment as the predicted price at that time step.

$$\begin{aligned} f(x_1) &= \mathbf{p1_1} \\ f(x_1, x_2) &= p1_2, \boldsymbol{p2_2} \\ f(x_1, x_2, x_3) &= p1_3, p2_3, \boldsymbol{p3_3} \\ f(x_1, ..., x_n) &= p1_n, ..., \boldsymbol{pn_n} \end{aligned}$$

$$\begin{aligned} \text{where} f &= \text{The bidirectional model} \\ x_i &= \text{The features at time step } i \\ pj_i &= \text{The predicted price at time step } j\text{, in the } i\text{th increment} \end{aligned}$$

The final time series of predicted prices is:

$$\boldsymbol{p} = \boldsymbol{p1_1}, \boldsymbol{p2_2}, \boldsymbol{p3_3}, ..., \boldsymbol{pn_n}$$

6.2.8 Context module

The context module was implemented using the following layers, all provided by Keras:

- Input layer: *keras.layers.Input.*
- State generation layer: *keras.layers.Embedding*.
- State reshape layer: *keras.layers.Reshape*

Figure 25 illustrates how the layers mentioned above are connected in the implementation.

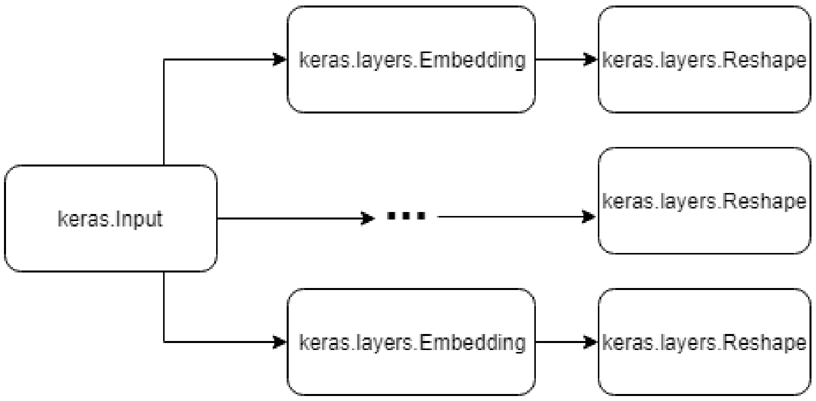


Figure 25: Connections between layers in the context module. The dots represent that varying amounts of State generation layers in parallel are possible

This module is responsible for generating the initial states for the LSTM-based models. The module takes in an input representing the context, for instance the digit representing the ID of a stock as done in this project, and outputs all initial states for the LSTM and bidirectional LSTM. As each layer of the vanilla and stacked LSTMs has two internal states - the hidden state and the cell state, this module produces two states for each layer. For the bidirectional model, this module produces four states for each layer, as a bidirectional layer consists of two LSTM layers. The embedding layer processes the input representing the context and generates an output that corresponds to this input. The state reshape layer is then used to make the state generated compatible with the state input of the different LSTM models.

6.2.9 How the LSTM models are combined with the context module

The vanilla/stacked LSTM models are combined with the context module using the layers described in Section 6.2.6 and Section 6.2.8. Figure 26 illustrates how the Keras layers are connected in the implementation.

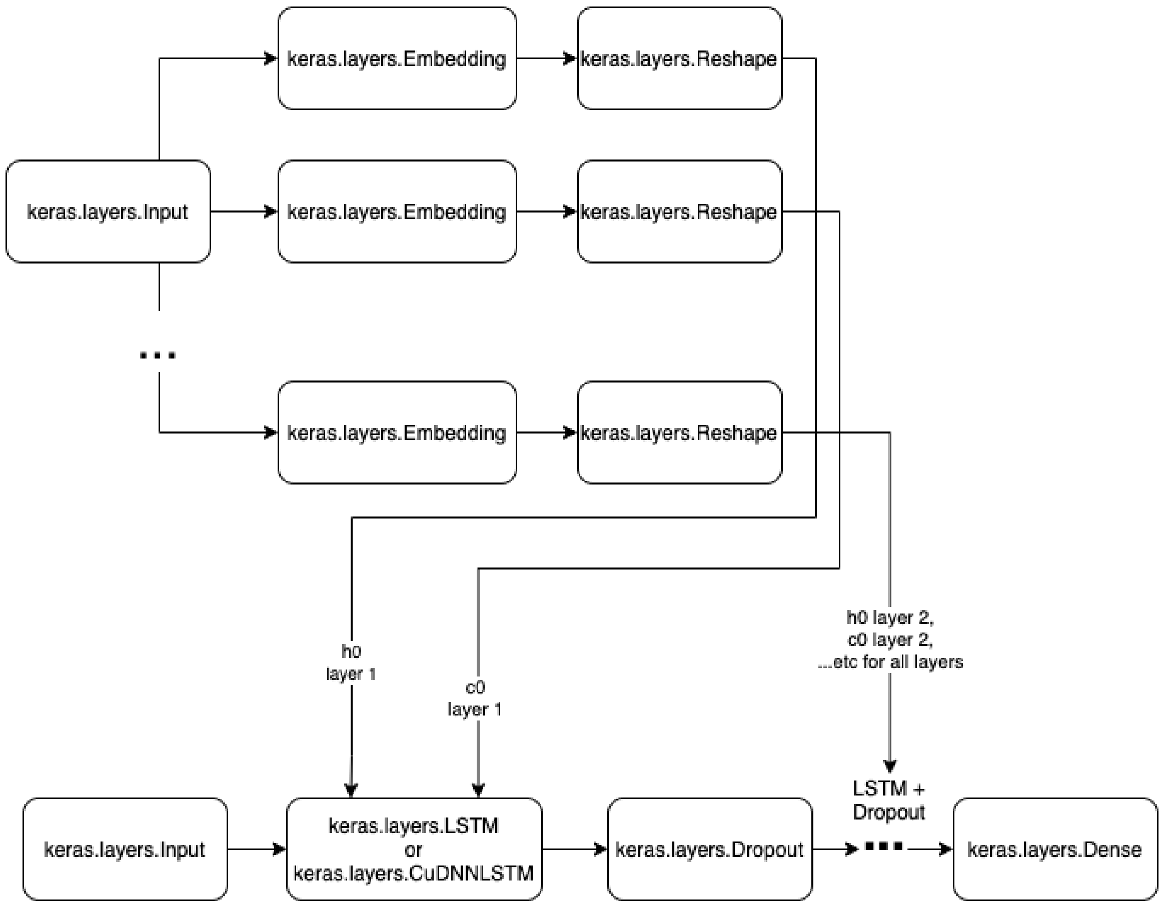


Figure 26: Implementation of the context module in combination with the vanilla or stacked LSTM models

The top Input layer in Figure 26 receives the stock-representations while the bottom Input layer receives the input features (trading data, sentiment data and trendscore data). The two top Reshape layers will send the output into the first LSTM layer as the initial states (hidden state and cell state). The next two Reshape layers (only one is shown in the Figure) will pass the output to the next LSTM layer, etc.

The bidirectional LSTM model was combined with the context module using the layers described in Section 6.2.7 and Section 6.2.8. Figure 27 illustrates how the Keras layers are connected in the implementation.

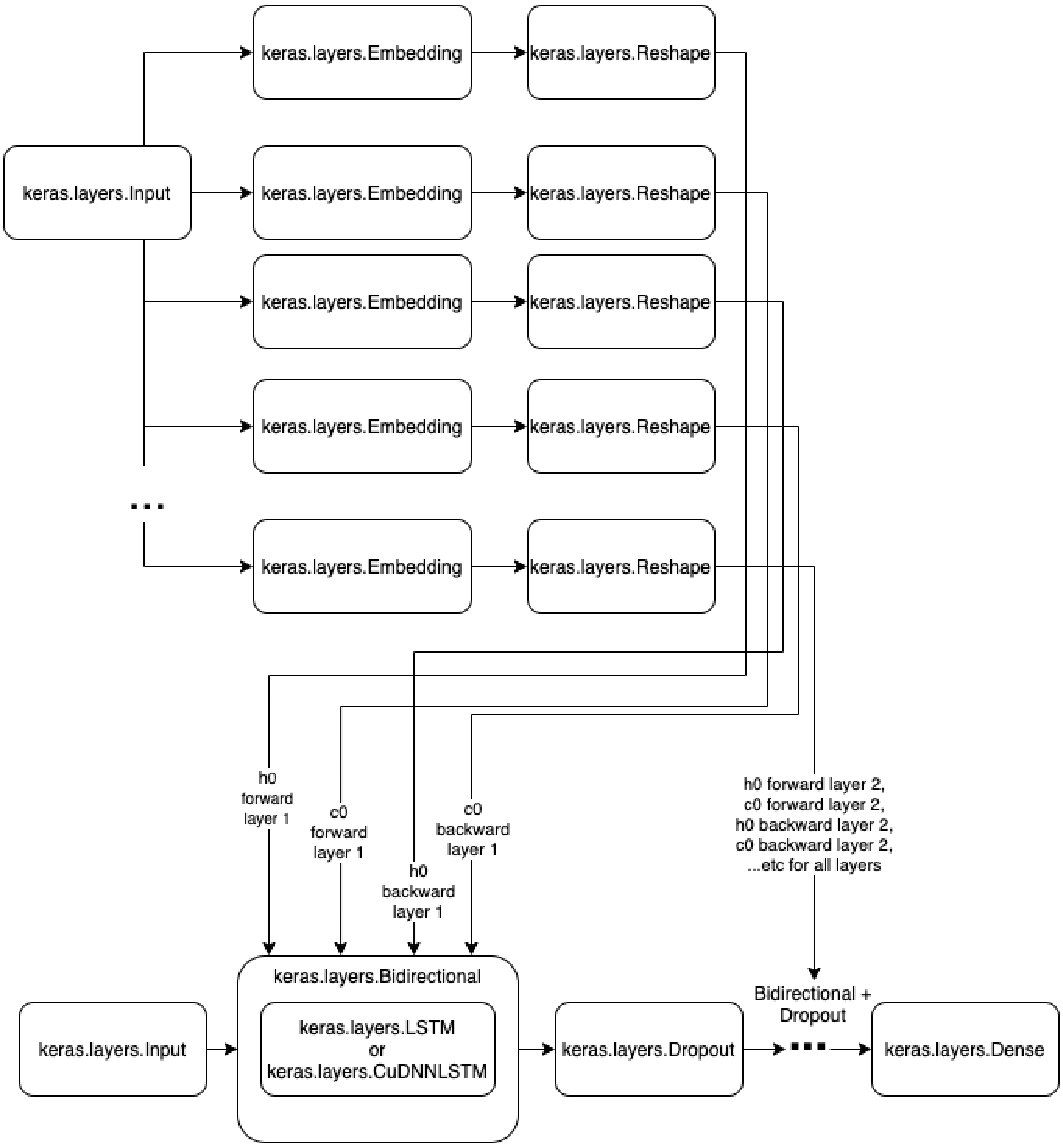


Figure 27: Implementation of the context module in combination with the Bidirectional LSTM model

The top Input layer in Figure 27 receives the stock-representations while the bottom Input layer receives the input features (trading data, sentiment data and trendscore data). The four top Reshape layers will send the output into the first Bidirectional layer as the initial states (hidden state and cell state for the forward layer, and the hidden state and cell state for the backward layer). The next four Reshape layers (only one is shown in Figure 27) will pass the output to the next Bidirectional layer, etc.

6.3 Evaluation metrics rationale and implementation

There are several evaluation metrics that are used across a multitude of works in the literature. In order to be able to compare results with other works, in addition to be able to compare internally with the implemented models, it is a good idea to implement evaluation metrics that facilitate this. MAPE is one of these widely used metrics, such as in Bollen, Mao & Zeng (2011) and Xiong, Nichols & Shen (2016), and should be considered. Additionally, MSE and MAE have been widely used in many related regression tasks, such as in the works of Bouktif, Fiaz, Ouni & Serhani (2018) and Næss (2018), and will therefore be included as metrics of which performance will be evaluated.

In addition to the commonly used metrics for regression, a last metric, direction accuracy (DA), will also be implemented. Direction accuracy is included to provide a different way to evaluate the results, and reflects an aspect important when dealing with financial tasks that is not necessarily captured by the usage of metrics only focused on evaluating regression. This project aims not to improve the direction accuracy on related tasks, but implements it as a means of investigation and discussion whether a model learning regression can be improved in terms of a metric that can result in tremendous results in practical use.

All of these relevant evaluation metrics are described in Section 2.1.16.

Implementation

MSE

MSE was computed using *sklearn.metrics.mean_squared_error*, the MSE metric provided by Scikit-Learn.

MAE

MAE was computed using *sklearn.metrics.mean_absolute_error*, the MAE metric provided by Scikit-Learn.

MAPE

MAPE was computed using Equation 10 defined in Section 2.1.16. The absolute value was computed using *numpy.abs*, and the mean value was computed using *numpy.mean*, both provided by the NumPy package.

DA

For the direction accuracy, the following method was used:

1. For both the predicted values and the actual values, moving one day at a time: give the value 0 if the actual value of the current day is higher than the next day, give it 1 otherwise. The result is two lists that represent the direction of predicted values and actual values.
2. Compute the accuracy between these lists.

7 Experimental plan

This chapter will present a thorough description on how the experiments were carried out and outlines the experimental plan that was followed. This includes the explanation and motivation on why the experiments were conducted in this fashion, the procedure that was followed. Elaboration on the hyperparameter search and the feature search parts is also present. Description on how the results were evaluated is also included in this chapter.

7.1 The essential parts of the experiments

The objective of the experiments is to generate data as a basis for answering the research questions. These are:

I) **Are there some patterns in the trading data, sentiment data and search popularity data gathered in this project that can facilitate price prediction?**

II) **Will a model based on LSTMs outperform the baseline models in predicting the next day prices of stocks?**

III) **Will introducing a context module improve prediction?**

IV) **Can one of the models with configurable parameters in this project outperform the baseline models in predicting stock prices, using the same set of parameters for every stock in this project?**

To answer research question I), the experiments comprise of implementing and optimizing vanilla LSTM models, stacked LSTM models and bidirectional LSTM models. These models will then be used to predict prices using different combinations of features. In order to answer research question III) a context module is added to some of the models that seem promising, in order to compare the context-free model with a model with a context.

In addition to the implemented LSTM models, implementation of several baseline models was also conducted. The baseline models primarily act as points of reference to put the results into perspective, and enable discussion and comparison with models that are based on different theories and ideas, as well as different complexities. This analysis enables answering research question II), as well as giving further insight on research question I).

Answering research question IV) meant that the models needed to be restricted in terms of parameter/hyperparameter tuning. By performing hyperparameter search for the LSTM models, a set of favorable hyperparameters that would work on all stocks could be determined before evaluating on the test set.

Overall, the tasks necessary can be summed up into several parts: Hyperparameter search, feature search, prediction on the test set, and evaluation.

7.2 Experiments outline

The experimentation evolved over time due to new evidence appearing as the experiments went on. To make sense of the expansions and additional experiments, an outline of the conducted experiments will be described here, including the reasoning behind the experiments that were added as a consequence of the initial results. This section will also work as a supplement that gives prior knowledge to the order of the results presented in Chapter 8.

Note that since the experiments ran on GPUs (Graphic Processing Unit), the results were non-deterministic due to the nature of how GPUs compute the gradients. It practice, this means that it was not possible to reproduce the exact same result in a deterministic way, for instance by setting the seed.

7.2.1 Initial experiments

Experimenting with baseline models

The experimenting started with producing results with the baseline models. As the baseline models did not undergo any hyperparameter search, the data were only divided into two parts, training data and test data. The models were trained on the training data, and evaluated with the test data.

Due to the ridge regression model having a parameter that could be tweaked, further experimenting were conducted to achieve better understanding of the task and the results. Also, an additional model that behaves closely to random guessing was implemented, to put the performance of the other models into perspective and to quantify what is meant by performance that is similar to random guessing.

Hyperparameter search

Before running experiments to produce results with the LSTM models, hyperparameters had to be determined. This consisted of determining the number of epochs to use as a guide first, followed by the actual hyperparameter searching. Due to time restraints, some restrictions were introduced to make this part manageable. Hyperparameter search is elaborated further in Section 7.3.1.

In order to decide the number of epochs related to the fitting of the models, the loss history graph was analysed. A fully fledged loss graph should tell the scale of the number of epochs before reaching the minimum validation loss. Are 10 runs sufficient, or 100, 1000, 10000, etc.? Also, the graph should reveal when the model starts overfitting, i.e. when the validation loss is starting to increase while the training loss is still decreasing. An arbitrary set of hyperparameter values was chosen for this analysis. The values chosen in this project produced a single layer LSTM with 160 hidden units, 0.2 dropout rate and MAE loss function. All features, elaborated in Section 7.3.3, were used.

The actual hyperparameter search used a single LSTM model with all features. The resulting hyperparameters were used in the subsequent parts.

Feature search with LSTM models

After the hyperparameters were determined, feature search (elaborated in Section 7.3.3), also called future analysis, was conducted. The first part of this experiment utilized the same data set as the hyperparameter search to give a basis for analysis and feature selection. The second part consisted of evaluation of the models on a set of features selected based on the results and analysis made in the first part. This part utilized the test set for evaluation, and these results were to be compared against results of the baseline models.

Including the context module

Following the process described above, a selection of the best LSTM configurations was identified and expanded upon using the context module. This was to see whether further improvements could be made. The prior trained models had a context module attached and was trained further, before being evaluated.

When training the model with context module, the process was similar to training the model without, only that the context module was trained simultaneously with the LSTM model. This would lead to the context module embeddings being modified together with the weights of the LSTM model being modified by training. In essence, the embedding layer used as the context module should have been optimized based on the feedback that was received during training.

7.2.2 Additional experiments due to the initial results

Experimenting with price change instead of price

Due to unfavorable results from the initial experiments, suspected being due to predicting the next day price with the current day price provided as a feature, experiments were expanded with predicting the price change instead. The experiment was ran on a single layered LSTM model with the hyperparameters achieved from the initial experimentation.

Because of some difficulties with calculating the MAPE, some adjustments were implemented. When calculating MAPE, one has to keep in mind that it is not defined when any of the true values are zero. The MAPE also yields very high values when true values are close to zero. This can be seen in the Formula 10. The results can in these cases be hard to analyse in a meaningful way. It was the case that there were some true price changes that were zero, making it a challenge to use the output as is. To circumvent this issue, the predicted price changes were transformed to predicted next day prices by adding the predicted price change to the real current price for each time step.

Experimenting with lagged variables

Another approach investigated in an effort to mitigate the issues present was introducing lagged variables, mainly with a model predicting the change of price, although some short experimentation were done on a model predicting price as well. These experiments were conducted with a single layered LSTM model and the same hyperparameters as the other experiments.

Simplifying the scope by analyzing one stock at a time

As the modifications mentioned above did not yield substantial performance improvements, the scope of the project was reduced considerably in order to attain understanding of the issues present. This was done by analyzing one arbitrarily chosen stock and performing experiments on this particular stock only. Additionally, this was done to investigate whether the issues stemmed from trying to predict a relative large set of different stocks, or had its origin on a more basic level. With some promising initial results, the experimenting was extended to analyze several additional stocks individually.

7.2.3 Evaluation on the test set

During the experiments, all evaluations were conducted on the validation set, to give the freedom of experimentation while still maintaining the integrity of the results on the test set. The evaluation on the test set was conducted after all the experiments were concluded, with a selection of configurations based on the results and observations from the experiments. This process was based on both the time constraint and the idea that configurations with inadequate results on the validation set are not interesting to investigate further due to already having displayed lacking results. Even if the performance on the test set might be better, evidence of inadequate performance already exist. Some results that are deemed lacking on a more general level are still included, in order to have enough evaluation on the test set to answer the research questions.

7.3 Elaboration

7.3.1 Hyperparameter search

The hyperparameters that can be configured in order to optimize the model are shown in Table 4. Some of these hyperparameters are continuous values while others have a high number of different possible discrete values. Testing all possible values for each of the hyperparameters would neither be beneficial nor feasible. Given that all hyperparameters have n possible values, the number of different models possible to be generated would be n^7. Due to time constraints, this number would quickly be unmanageable when n increases. Even with only $n = 3$, $3^7 = 2187$ models would have to be created in order to test and analyze all different values. Instead, some hyperparameters were prioritized, which would be the hyperparameters that are the most specific to the data at hand. These hyperparameters will have different potential values in the search. The values of other hyperparameters will be based on conventional values in the literature. In this experiment, the hyperparameter search consists of a search through all the different values to try in Table 4.

Hyperparameter	**Values**	**Justification**
Sum of number of hidden units in the LSTM	$32, 128, 160$	A wide range of numbers of hidden units
Learning rate	0.001	Default value for Adam
Epochs	Analyse training and loss graph to decide	This hyperparameter is highly specific to the task at hand. The graph will therefore be analyzed to decide the value to use
Dropout rate	$0, 0.2, 0.5$	According to Goodfellow, Bengio & Courville (2016, p. 259), 0.5 is recommended for hidden layers, 0.2 for the input layer. A 0 dropout will also be analyzed in order to examine whether dropout will improve the model.
Optimization method	Adam	According to Brownlee, 2020 “Adam is relatively easy to configure where the default configuration parameters do well on most problems”.
Batch size	15	This is the amount of stocks included in the project
Loss function	MSE, MAE	These are sensible when predicting prices using regression

Table 4: Hyperparameters used in hyperparamater search

To evaluate which hyperparameters are optimal, comparisons over different metrics are used - MAE, MSE, MAPE and DA. As these are four different dimensions, some set of hyperparameters might be better than some other set in some aspects. For instance, one model might be superior in MSE but inferior in MAPE in relation to another model. Therefore, deciding the set of hyperparameters to continue with requires some thought process when the results are clear. The process utilized in this project is described in Section 7.4.

7.3.2 Multiple layers and bidirectional layers

Experiments involving the stacked LSTM or the bidirectional LSTM will be conducted in the same manner as for the vanilla LSTM. The only difference is that the sum of number of hidden units in the LSTM (see table 4) will be distributed through the different layers. For the stacked LSTM, two and three layers will be implemented and analyzed where each layer have approximately the same amount of hidden units. The bidirectional LSTM was planned with a similar implementation, but due to the initial results and analysis, further exploration of this model was left out.

7.3.3 Feature analysis and feature subsets

Data type	**Consists of**
Trading data	[Price], [Price change, Direction], [Volume], [Open, High, Close]
Sentiment data	[Positive, Negative, Neutral], [Positive proportion, Negative proportion, Neutral proportion]
Trend data	[Trendscore]

Table 5: Data categories divided into feature subsets - described in section 4.2

The objective of the feature analysis is to identify relationships between features and future prices to be predicted. All of the features available to search through are presented in Table 5.

Data types	**Feature subset**
Price	{Price}
Price + Trading data	{Price, Price change, Direction, Volume, Open, High, Close, Direction}
Price + Sentiment data	{Price, Positive, Negative, Neutral, Positive proportion, Negative proportion, Neutral proportion}
Price + Trend data	{Price, Trendscore}
All data	{Price, Price change, Direction, Volume, Open, High, Close, Direction, Positive, Negative, Neutral, Positive proportion, Negative proportion, Neutral proportion, Trendscore}

Table 6: Feature subsets

The number of subsets of a set of size n is $2^n - 1$, excluding the empty set. The set of all features can therefore be divided into $2^{13} - 1 = 8191$ subsets. Given the time restrictive nature of the project, training and analysing 8191 different feature subsets is infeasible. The features were grouped together based on how strongly related they were perceived to be, in order to reduce the number of feature subsets. E.g. Positive proportion, Negative proportion and Neutral proportion are features that are highly related to each other and was grouped together. The feature arrangements are presented in Table 6. The price will be included in every group as predicting the next price without having the current price can be very hard in some cases. For instance, when only “trendscore” is present, predicting the next price would be extremely hard. If some of the feature subsets yield better results than only price, these will be combined in order to analyse whether a combination of these will yield even better performance.

7.4 Evaluation

Ranking the models

To evaluate a model, MAE, MSE, MAPE and DA were used. As the performance of a model in relation to these metrics can vary depending on the run, in this experiment, every configuration of a model or features was run three times, and the scores were averaged. The discussion and analysis were, however, based on both the averaged score of the runs as well as the score of each individual run.

In the feature subset analysis and hyperparameter search steps of the experiment, a pool of different models is generated. These will be presented in a table along with their rank in the different metrics, ordered by the sum of their rank over the different metrics. After the hyperparameter search only the superior configuration of hyperparameters was retained, i.e. the configuration that yielded the lowest sum of ranking scores. A similar procedure followed in the feature search, although not with the same restrictive manner as investigating both features that resulted in an improved performance and worse performance could be interesting.

8 Results

The results with accompanying analysis and comments are presented in this chapter. This content has the aim to produce material for discussion in Chapter 9, and to answer the research questions motivating this project. This chapter starts with the baseline results, to give a point of reference for comparison. Experiments to determine hyperparameters and training regime follows, with experimenting on the LSTM models after that. Experimenting with the LSTM models are the main focus of thesis, so it was allocated the largest amount of available time. The experimentation can be divided into two major parts. In the first part, the LSTM models are configured to predict the next day price (Section 8.4, 8.5, 8.6 and 8.7), while predicting the price change was the main focus of the second part (Section 8.8, 8.9 and 8.10).

Color coding in tables

Color coding has been used in tables to convey additional information and to differentiate the information displayed. The following colors and their meaning have been used:

- **Red**: Results generated from the baseline model
- **Yellow**: Results extracted from a previous table
- **Green**: Results generated on evaluation on the test set

8.1 Baseline models

Implementing the baseline models has the main purpose to put the results gathered from experimenting with the LSTM models into perspective. Three models were chosen initially, with an increasing complexity between them to see whether this increase would yield positive results. Specifics on how the models were implemented are mentioned in Section 6.2.1, 6.2.2, 6.2.3 and 6.2.4.

The simple 1-step-behind model uses only price as input, while the linear regression model and ridge regression model both have all data as input. The results from these models are presented in Table 7. Since neither the linear regression model nor the ridge regression model showed any improvements over the 1-step-behind model, only the 1-step-behind model was used in comparison against the LSTM models. The 1-step-behind model is also referred to as the baseline model and the naive model from Section 8.5 and onward.

Model	MAPE	MAE	MSE	DA
1-step-behind	1.6567%	4.4283	143.2291	51.47%
Linear regression	1.6676%	4.4545	143.6266	49.94%
Ridge regression	1.7718%	4.8482	158.6432	49.17%

Table 7: The baseline models on the test set

Table 8 presents the performance of the naive model measured on the validation set, as this model was compared against the LSTM models the most in the later sections, due to this model having the best performance of all the baseline models.

Model	MAPE	MAE	MSE	DA
1-step-behind	1.342%	3.153	52.42	52.61%

Table 8: The naive model on the validation set. The purpose of this table to for compare the different LSTM models that have only been measured on the validation and not yet the test set

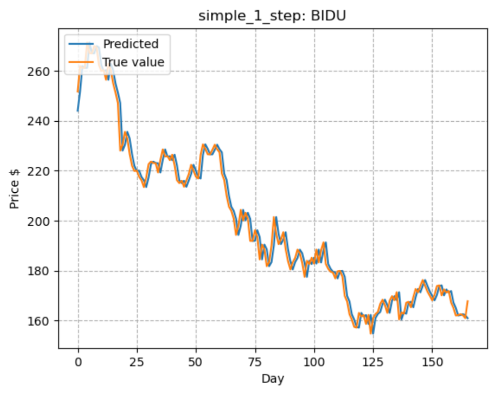
Prior to analyzing the results of the baseline model, a random guessing model was implemented. This model and the result is presented in the next section, Section 8.1.1.

8.1.1 Evaluating DA when random guessing

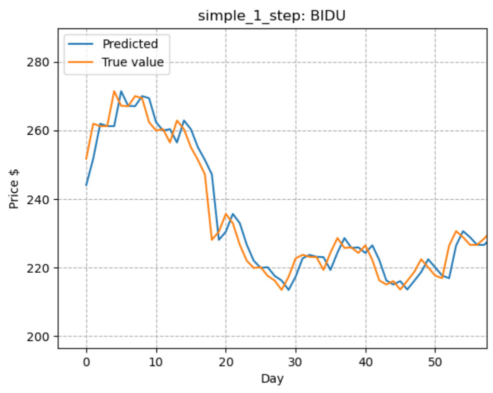
To put into perspective how the models perform in terms of direction accuracy compared to random guessing, the baseline experiments were extended with a model that performs closely to random predictions. This random guessing model takes the current price as an input and outputs a price that has an equal chance of being higher than the current price as well as being lower. The model was run 1000 times, and the DA averaged over these runs, which resulted in a DA score of 49.99%. This result is as expected since, intuitively, the probability of guessing something correct when there are two choices is 50%, given that the process deciding the actual value is random too. With the DA score of the random guessing model being close to 50%, this indicates a random process, or at least a process close to a random process, that decides the day to day direction of stock prices. Although the general market value has risen over time, suggesting the process to not be entirely random, when examining a limited time frame such as the test set, the process still resembles a random process, implied by the DA score of this experiment.

8.1.2 Simple 1-step-behind model

Inspecting the regression scores MAPE, MAE and MSE together with the resulting plotting of the results, such as the one presented in Figure 28a, the simple 1-step-behind model gives the impression of doing a good job at predicting the stock prices. Investigating the plots close, however, reveals the opposite. Figure 28b shows the results from Figure 28a zoomed in on the 50 first predictions. This highlights what can be a problematic issue with using such a model. With such results, financial gains in the stock market are hard, as the model always is one step behind the actual prices. This is also reflected by the DA score, as the model performs close to what can be considered as guessing the direction of the stock prices. Discussion on how this is problematic is included in Chapter 9.



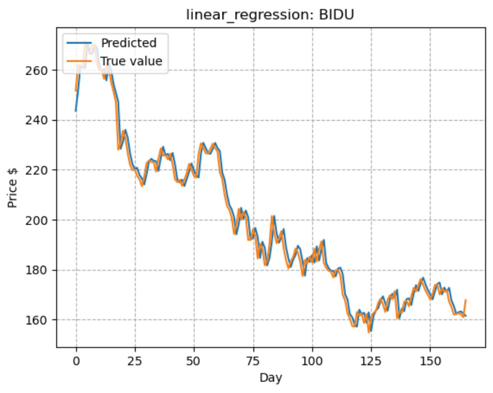
(a) Results for the stock BIDU



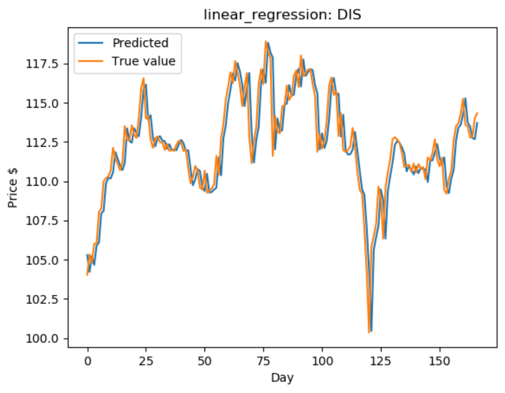
(b) Results zoomed in on the first 50 predictions

Figure 28: The simple 1-step-behind model predictions on the test set for BIDU

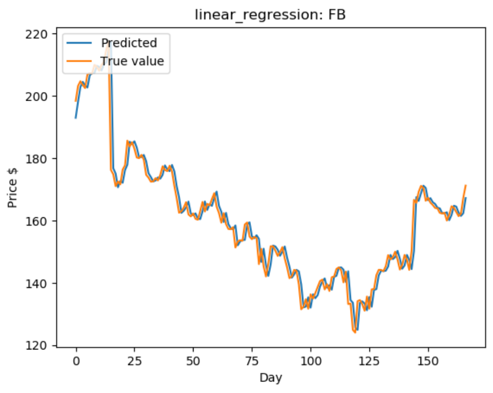
8.1.3 Linear regression



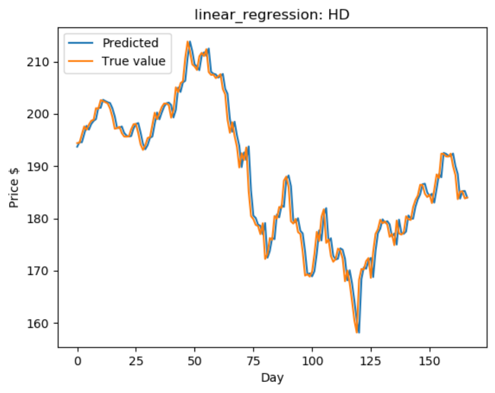
(a) Bidu



(b) DIS

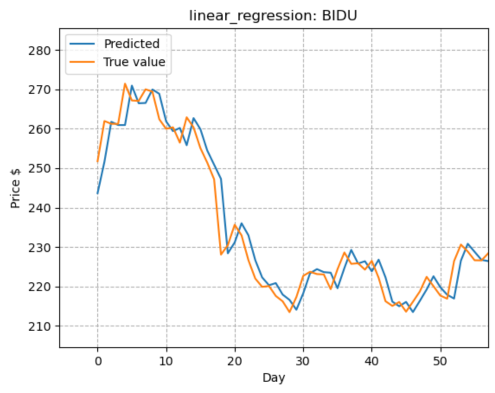


(c) FB

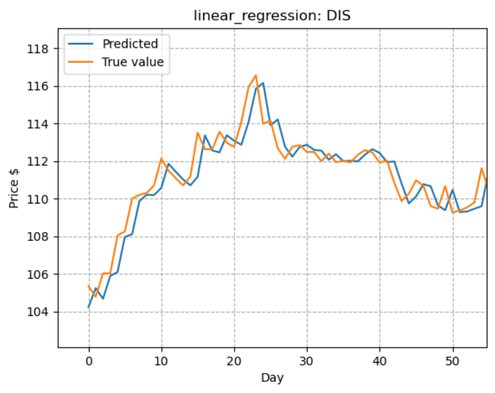


(d) HD

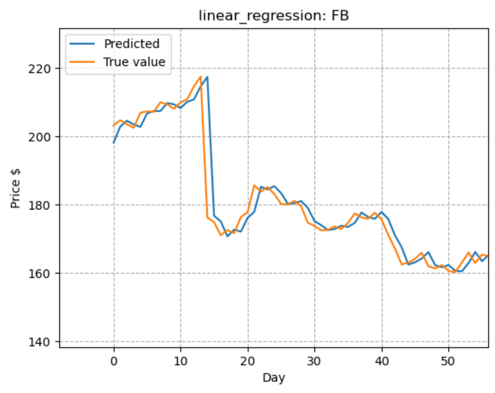
Figure 29: The linear regression model predictions on test set for the stocks BIDU, DIS, FB and HD



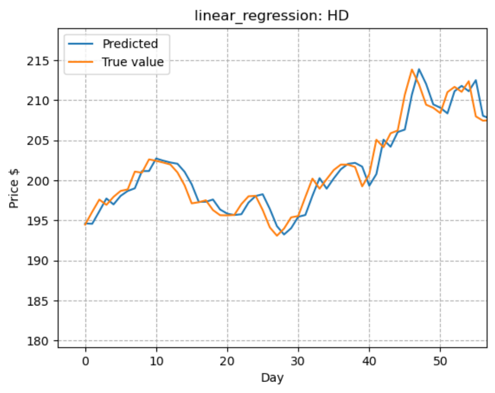
(a) BIDU



(b) DIS



(c) FB



(d) HD

Figure 30: The linear regression model predictions on test set for the stocks BIDU, DIS, FB and HD, zoomed in on the 50 first predictions

The regression scores achieved by the linear regression model seem to be relatively good when inspecting them briefly, which is also supported by the plots as illustrated in Figure 30. However, when examining the plots closer, they exhibit the same characteristics as the plots from the simple 1-step-behind model. As can be observed from the plots in Figure 30, all the results seem to consistently be one step behind the actual value, as if the linear regression model has learned that only the current price should be considered when predicting the price of the next day. Despite the relatively low regression error, displaying this kind of characteristic makes it hard for real life usage. Similar to the simple 1-step-behind model, this results in the linear regression model achieving relatively low DA scores. These results are, however, expected. The correlation analysis in Section 4.3 indicated that the next day price only exhibited linear correlation with related trading data, such as the current day closing price and open price.

8.1.4 Ridge regression

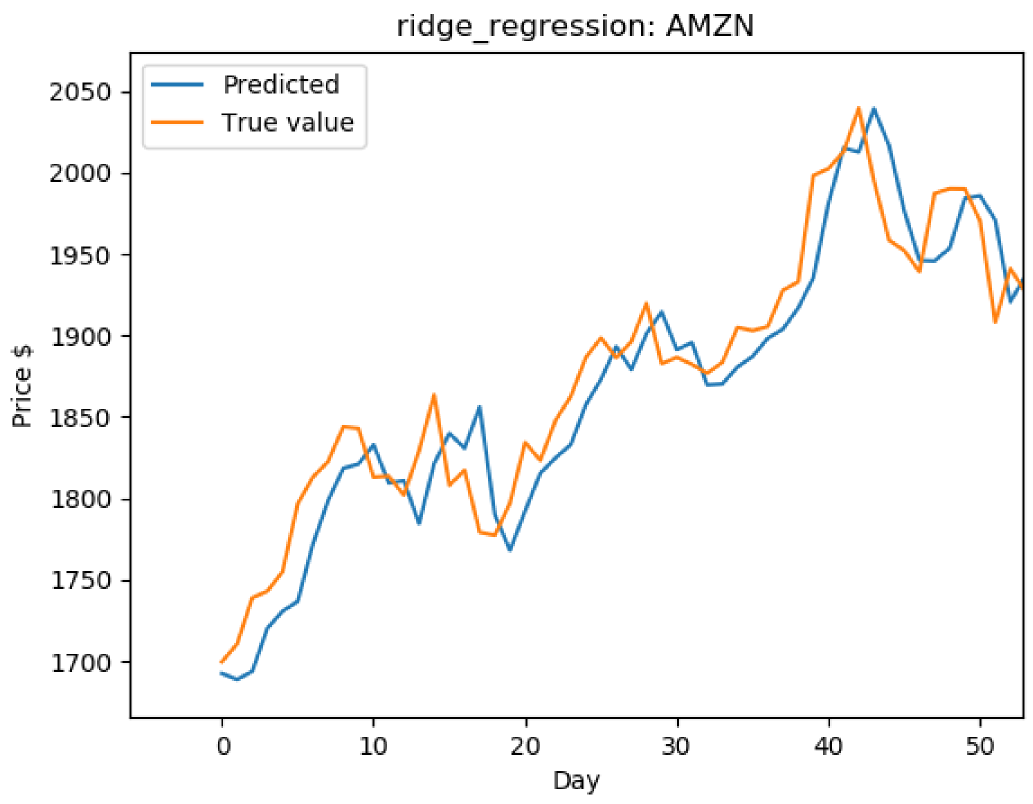


Figure 31: The ridge regression model predictions on test set for AMZN, zoomed in on the first 50 predictions

Examining the results and the plots of the results, it seems that the ridge regression model consistently achieves worse results than the previous two models. The model also seems to exhibit the same issues as the other models (illustrated in Figure 31), mainly that it is overly dependent on the price related features, such as price, open, high and low. This is not unexpected, as the other features do not have high correlation with the next day price. If there are any relationships between the output to be predicted and the other features, a model able to capture non-linear relationship is necessary. The ridge regression model was inspected further in Section 8.1.5 as an effort to identify the reasons for the results achieved, and to obtain better understanding of the input features effect on prediction.

8.1.5 Extending the baseline experiments

Some investigative experiments were commenced as a consequence of the obtained results above. The investigative experiments were used to gain better comprehension of the achieved results. It should be noted that the experiments were done on the actual test data, and should be treated with this in mind.

Examining ridge regression further

As the ridge regression model displayed results that were worse than the other baseline models, this model was examined further. This was done primarily by changing two different factors separately.

First off was experimenting with different α values to see how this would affect performance. By applying different values for α, information on how inclusion on other features was affecting performance could become more prevalent.

α	**MAPE**	**MAE**	**MSE**	**DA**
2.0	1.8123%	4.9920	165.8314	48.81%
1.0	1.7718%	4.8482	158.6432	49.17%
0.5	1.7507%	4.7817	155.3082	50.42%
0.2	1.7264%	4.7129	152.3614	50.87%
0.1	1.7084%	4.6571	149.8386	50.91%
0.0	1.6671%	4.4534	143.6018	49.86%

Table 9: The ridge regression model with different values for α.

The results from Table 9 show that the performance gets worse as the α value grows. This indicates that the model is not able to make use of features other than those related to price. When being forced to take other features into consideration, the performance is affected negatively. When the α is set to 0.0 the performance is similar to the linear regression model (the discrepancy is due to the implementation of *sklearn.linear_model.Ridge*), as expected. As this is the best performing configuration, it indicates that forcing the model to take into consideration other features is not helpful, at least when dealing with a model capable of capturing linear relationships only.

The second factor to be varied was features. To see how they impacted results, one feature was left out at a time, a procedure also called ablation, producing the following results:

Omitted feature	**MAPE**	**MAE**	**MSE**	**DA**
price	1.8702%	5.1535	174.4701	49.17%
high	1.7898%	4.9019	160.2271	49.41%
low	1.7855%	4.9004	164.5560	48.97%
open	1.7420%	4.7449	154.3025	49.53%
volume	1.7699%	4.8437	158.7204	49.61%
direction	1.7999%	4.9338	162.0643	50.46%
neutral_prop	1.7717%	4.8481	158.6460	49.13%
positive_prop	1.7717%	4.8479	158.6125	49.13%
negative_prop	1.7717%	4.8480	158.6297	49.21%
negative	1.7717%	4.8452	158.3191	49.21%
positive	1.7718%	4.8476	158.6376	49.25%
neutral	1.7714%	4.8463	158.6257	49.05%
trendscore	1.7528%	4.7824	154.2285	50.67%

Table 10: The ridge regression model with different features omitted.

From the results presented in Table 10, it can be seen that there are some features that seem to influence the performance in a negative way. Thus, by omitting them, performance may increase. For instance, omitting the feature "trendscore" resulted in performance noticeable better than when including it. Investigating this further, however, indicated that this might not always be the case. When running the same setup with a different training and test split (80% and 10%, respectively), the results did not indicate similar noticeable improvements. Except for "price", there seems to overall not be any one feature that is the sole reason for a performance increase or performance degradation, when experimenting with the ridge regression model.

Model	MAPE	MAE	MSE	DA
1-step-behind	1.4858%	2.7978	15.4556	56.36%
Linear regression	1.4936%	2.8100	15.6733	53.33%
Ridge regression	1.4795%	2.7829	15.3712	58.18%

Table 11: The baseline models on the stock AAPL, ridge regression optimized

Additional investigations of α showed that better results could be achieved if the α was tweaked to optimize performance on each individual stock. For instance, when applying an optimized α when processing the AAPL stock only, performance in terms of all metrics exceeded the other baseline models, as depicted in Table 11. This suggests that it is possible to achieve better results with the ridge regression model comparable to the other baseline models, although the issue of constantly lagging behind is still present. Since optimizing α manually for each individual stock was deemed out of scope due to time limitations and research question IV), this was not investigated further. Chapter 9 will discuss the results achieved in Table 11 further, examining whether this approach is applicable on other time frames as well.

8.2 Analyzing the loss history

Understanding how long the LSTM models should be trained for is a challenging task. To get a sense on what should be done with regards to this matter, a run with a model configured to have 160 hidden units, 0.2 dropout and MAE as loss function was conducted with an exaggerated amount of epochs. The result is illustrated in Figure 32.

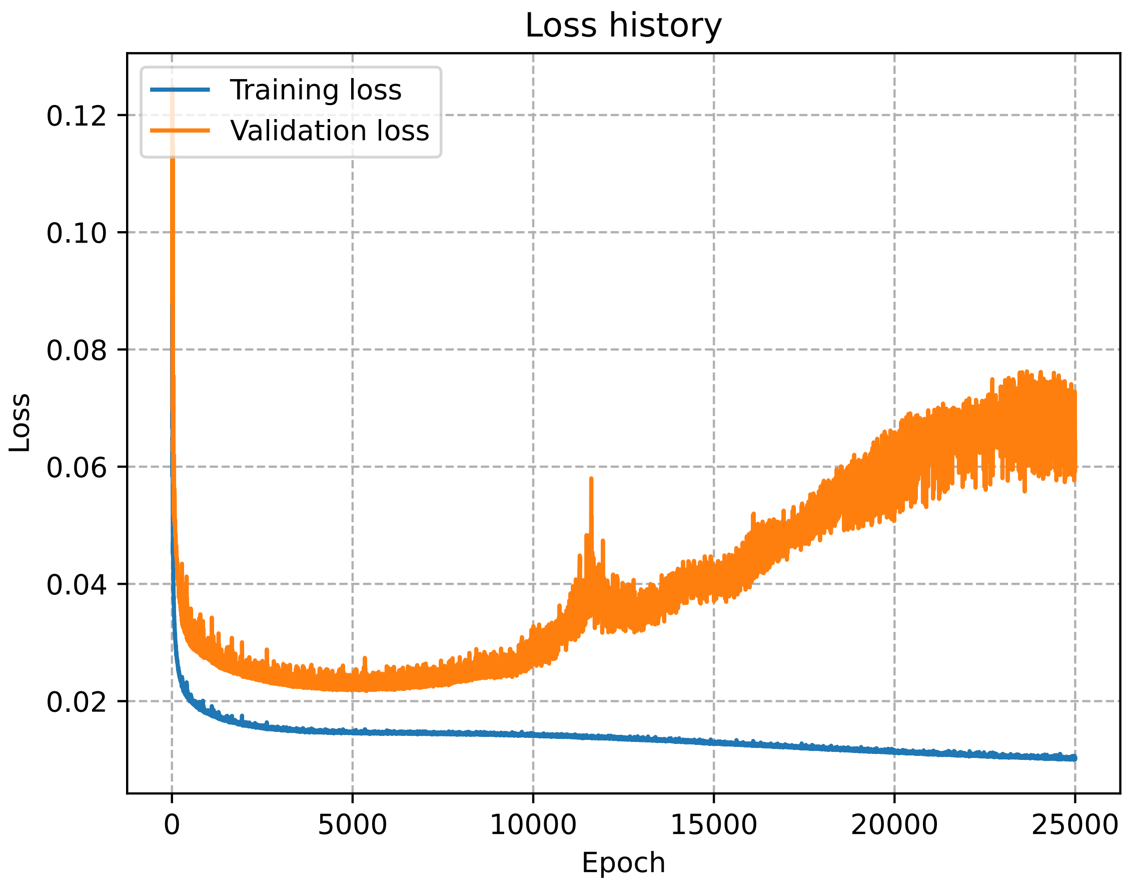


Figure 32: Training/validation loss during training using a single layered LSTM with 160 hidden units, 0.2 dropout rate and MAE loss over 25000 epochs. The first 10 epochs were removed from the graph because of high values compared to later values. Including these would make it harder to analyze the later values.

In Figure 32, the validation loss generally decreases until around epoch 5000 where it reaches the minimum. After this, the validation loss increases while the training loss continues decreasing. This is a classical sign of overfitting. As it is very hard to predetermine exactly which epoch is related to the lowest validation loss, early stopping will be used. Early stopping is a tool used for stopping the training prosess at a favorable time. It uses the “patience” parameter in order to decide when to stop. If the model has not improved on the validation set in the last x epochs (where x is the patience), the training process should stop. Early stopping will in addition to stopping at a good time also be used to save the weights related to the epoch where the loss was lowest in order to restore these at the end of training. Observing the graph, it seems that one would safely reach the minimum validation loss by using a patience of 1000. One should keep in mind that the loss graphs will be different for other hyperparameters and features. Even with that in mind, 1000 epochs for the patience seems fairly safe. Figure 33 shows that the model finds the minimum using early stopping with a patience of 1000 without running too many unnecessary epochs after reaching the minimum.

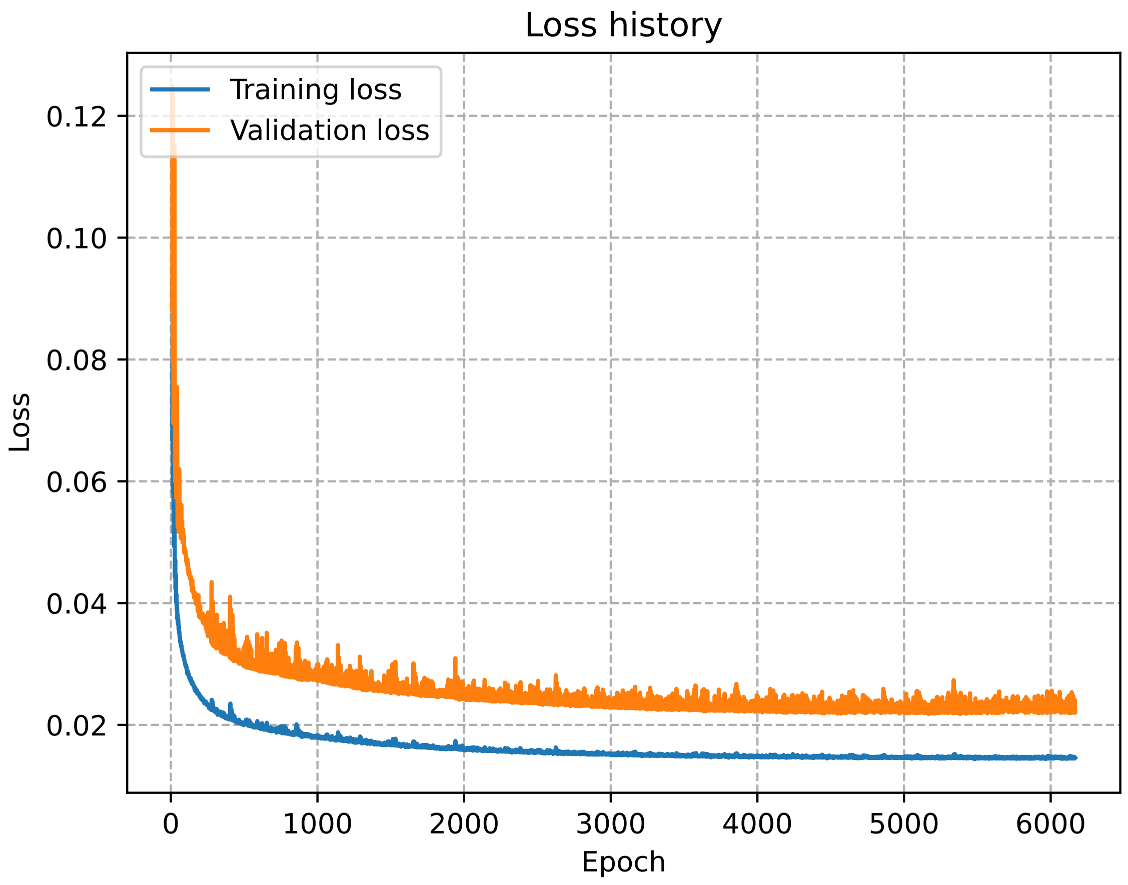


Figure 33: Using patience of 1000 epochs

8.3 Hyperparameter search

As described in Section 7.3.1, an experiment to determine what hyperparameter to use in the later experiments was conducted. The top three results and two bottom results are presented in Table 12.

Dropout	**Number of units**	**Loss function**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
0.0	[160]	mae	1.396% (#1)	3.314 (#1)	57.52 (#1)	50.76% (#4)
0.0	[128]	mae	1.403% (#2)	3.329 (#2)	58.34 (#2)	50.29% (#6)
0.2	[128]	mae	1.406% (#3)	3.553 (#3)	68.62 (#3)	50.06% (#10)
0.5	[32]	mae	1.768% (#17)	5.849 (#17)	224.0 (#17)	48.13% (#16)
0.5	[32]	mse	1.849% (#18)	6.268 (#18)	260.0 (#18)	47.91% (#17)

Table 12: Hyperparameter search, top three and bottom two results sorted on the sum of ranks. "#" in the Mean MAPE, Mean MAE, Mean MSE and Mean DA denotes the rank of the result. The model with the best mean MAPE will have MAPE #1, the model with the second best mean MAPE will have MAPE #2, etc. The table is sorted by the sum of ranks for every metric in each row. See all results in Table 57

From the results in Table 12, and from all the results in general, some observations can be made.

Observing the number of units, it seems that models with a higher number of hidden units perform better overall in terms of overall ranking, seeing that the top performer has 160 units while the next two have 128 units. This makes sense as increasing the number of hidden layers increases the ability to solve complex tasks. However, this increase can also increase the chance of overfitting, which should be kept in mind when analyzing the results in later parts of the experimentation.

Table 12 shows that the top two performing models have zero dropout rate, while the bottom two performing models have dropouts of 0.5. This suggests that in order to achieve the best scores in terms of the evaluation metrics, the dropout rate should be kept as low as possible. However, since 0.0 dropout increases the risk of overfitting on the training data, this should be kept in mind if this dropout rate is going to be used.

Lastly, in relation to the loss function, the top six performing models use MAE as can be seen in Table 57. Observing this table, the best mean MAE score that is reached when using MAE as a loss function is 3.314 while the best mean MAE score when using MSE as a loss function is 3.671. Even when observing the mean MSE scores, using MSE as loss function is inferior; the best mean MSE being 70.91, while it is 57.52 when using MAE as the loss function.

Overall, the results from the hyperparameter search seem to suggest that a model with lower dropout rate and a higher number of hidden units will result in the best performance in terms of the evaluation metrics. This was the basis of the model experimented with in the later parts. Specifically, a model with no dropout, the number of hidden units set to 160, and MAE as the loss function.

8.4 Issues with the bidirectional implementation

The bidirectional LSTM model was experimented with, but the exceptionally poor results, presented in Table 58 in the Appendix, indicated major flaws in the implementation. The issues seemed to originate from a configuration unfit for the bidirectional LSTM model. Since the bidirectional model has a layer that processes the data backwards, during training, the model has access to information on the price of the next day when predicting the price of the next day. This makes the model tune the weights as if the model will have access to this information from the future. When predicting following the process described in Section 6.2.7, the model does not receive future information that the weights have been tuned to focus on. The model is therefore unable to predict accurately, as the configuration has made it incapable of handling the situation at prediction time. A solution is to instead implement a bidirectional model using sliding window: for each time step to predict the next price, the current and previous x features, where x is the sliding window size, are used. The bidirectional model processes these current and past features both forward and backward in order to predict the next price. In this way, the model never retrieves information from the future. This approach is similar to the approach of Althelaya, El-Alfy & Mohammed (2018). Due to the time constraints, moving in this direction was decided against, and further experimentation with the bidirectional LSTM model was abandoned.

8.5 Feature analysis, predicting price

First and foremost, this section will present comparisons between the different models; vanilla LSTM with 160 hidden units and stacked LSTMs, one with two layers consisting of [80,80] hidden units and one with three layers consisting of [54,53,53] hidden units. The hyperparameters are chosen based on Section 8.3. Secondly, this section will present comparisons between the different feature subsets in order to locate any valuable information in relation to the research questions.

The results related to the models and all the selected feature subsets are presented in Table 13.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
1-step-behind		1.342%	3.153	52.42	52.61%
Vanilla [160]	price	1.352% (#1)	3.157 (#2)	52.52 (#2)	52.47% (#2)
Vanilla [160]	price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.353% (#2)	3.155 (#1)	52.63 (#3)	52.58% (#1)
Vanilla [160]	price, trendscore	1.354% (#3)	3.158 (#3)	52.18 (#1)	51.78% (#5)
Stacked [80, 80]	price, trendscore	1.382% (#6)	3.257 (#4)	54.87 (#4)	52.04% (#3)
Vanilla [160]	price, open, high, low, volume, direction, change	1.393% (#7)	3.307 (#7)	58.76 (#8)	51.04% (#11)
Stacked [54, 53, 53]	price	1.415% (#10)	3.498 (#10)	64.17 (#10)	51.73% (#6)
Vanilla [160]	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	1.396% (#8)	3.314 (#8)	57.52 (#7)	50.76% (#13)

Table 13: Feature search using vanilla LSTMs and stacked LSTMs. All results related to the vanilla LSTM are shown, and the best result related to each of the stacked LSTMs.

One observation is that there seems to be a general pattern related to the number of layers in the model. The top three models only have one layer with 160 units while the next three have two layers with 80 units in each layer. Of the bottom five results, as seen in Table 59 in the Appendix, four of them have three layers of sizes 54, 53 and 53. It is clear that in this experiment, the vanilla LSTM is superior to the stacked LSTMs. This indicates that the added complexity the models gain by adding layers is not beneficial.

Another observation is related to the metric results. It seems that MAPE, MAE and MSE are correlated. The rankings of these results for each feature subset are usually very close, e.g. the rankings of the top result's MAPE is #1, MAE is #2 and MSE is #2, and the trend seems to continue with the subsequent results. The DA ranking, however, does not seem to follow the same trend to the same degree as the three other metrics, but it still seems to be a connection there as overall better performance in MAPE, MAE and MSE resulted in a relative high rank in DA. This can be seen in the top four configurations, which have relative high ranks in the three first metrics, but also a relative high rank with regards to DA.

Model	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
1-step-behind		1.342%	3.153	52.42	52.61%
Vanilla [160]	price	1.351%	3.134	51.83	52.61%
Vanilla [160]	price, positive_prop, negative_prop, neutral_prop	1.343%	3.141	52.35	52.44%
Vanilla [160]	price, positive, negative, neutral	1.346%	3.139	52.15	52.0%
Vanilla [160]	price, volume, direction, change	1.344%	3.137	53.32	53.05%
Vanilla [160]	price, open, high, low	1.351%	3.191	54.75	51.19%

Table 14: Showing the best of 10 individual runs for the different feature subsets. The row with yellow background is extracted from Table 59.

With regards to feature subsets, there are some interesting observations that can be made. The vanilla LSTM using only price is superior to the vanilla LSTMs using price in addition to other features. This is an indication that the LSTM has been overfitted and is finding patterns in noise when looking at the other features. One solution is to add regularization. Regularization in the form of a dropout layer was added in the hyperparameter search in Section 8.3 and did not seem to improve the model. Another solution is to reduce the number of features. Several of the feature subsets contain relatively many features. Price + trading data contains [“price”, “open”, “high”, “low”, “volume”, “direction”, “change”], price + sentiment data contains [“price”, “positive”, “negative”, “neutral”, “positive_prop”, “negative_prop”, “neutral_prop”]. In order to reduce the risk of overfitting, it might be worth it to divide these feature subsets into smaller subsets, such as [“price”, “open”, “high”, “low”], [“price”, “volume”], [“price”, “change”, “direction”], [“price”, “positive”, “negative", "neutral"] and ["price", "positive_prop", "negative_prop", "neutral_prop"]. Dividing these into even smaller feature subsets does not seem to be feasible in terms of time and this was therefore not explored further. Table 14 shows the results when dividing the feature subsets in the described way, using a LSTM with one layer and 160 hidden units, as this configuration was the one showing most potential. The table presents the best results from 10 different runs, contrary to Table 13 where the average of 3 runs was shown. Even when dividing the larger feature sets into smaller subsets, the results in Table 14 show that no model was clearly superior to the model using only "price".

In Table 59 in the Appendix it can be seen that having "trendscore" in addition to "price" yielded MAE and MSE score superior to the baseline model in two of the three runs. This is an interesting case, as experimentation with the ridge regression model in Section 8.1.5 suggested that including "trendscore" lead to worse performance. The results on the LSTM model indicate that "trendscore" can improve performance, and that the relationship might be non-linear. Chapter 9 will discuss this matter further. Because of the results, "trendscore" in addition to price was examined closer. These two subsets were run 10 times each and the best results are presented in Table 15.

Model	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
1-step-behind		1.342%	3.153	52.42	52.61%
Vanilla [160]	price, trendscore	1.347%	3.119	51.15	53.9%
Vanilla [160]	price	1.346%	3.129	52.66	52.57%

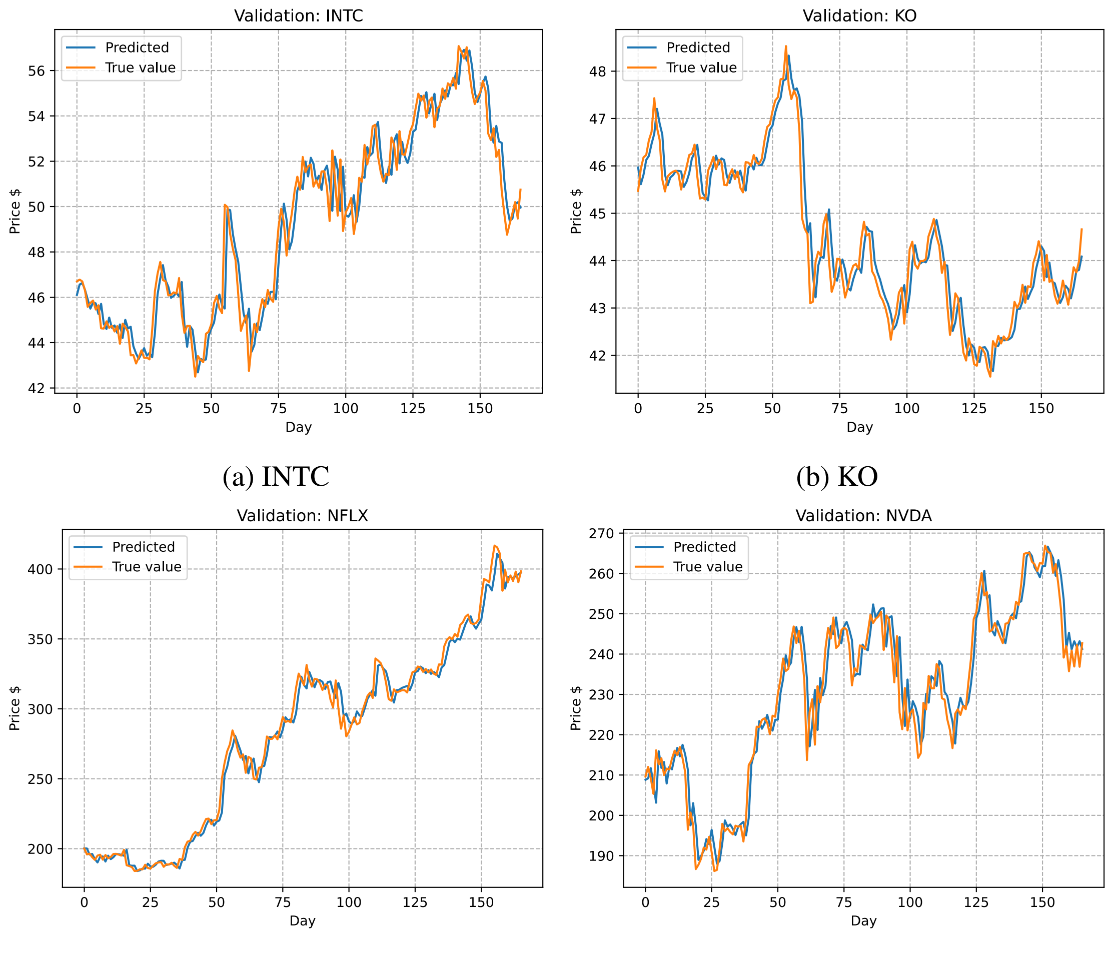
Table 15: Optimising model related to each feature subset 10 times. The table shows the best individual run for each feature subset.

Table 15 shows that adding "trendscore" to "price" yielded slightly better results. This is an indication that "trendscore" does give some valuable information in relation to stock price prediction. On the other hand, it might also be because of random variations as a result of the random nature of LSTMs. It is possible that running the same experiment on only "price" for 100 times would yield better results. This is based on the observation that having "trendscore" in addition to "price" does not consistently result in greater performance compared to using "price" alone.

Comparing the predicted prices to the actual prices

To analyse the performance of the LSTM models on this task the plots generated by the results were analysed closer, which revealed some points that will be described in the upcoming paragraphs and discussed further in Chapter 9.

When inspecting the graphs initially, it seems like all models are able to generate predictions that are close to the actual values. This is illustrated in Figure 34. However, inspecting the graphs closer reveals that this is not true.

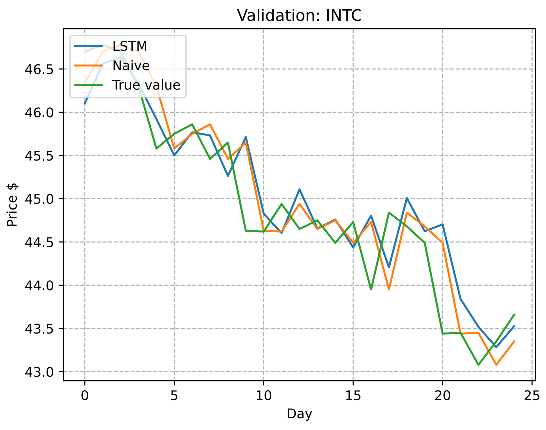


(a) INTC (b) KO

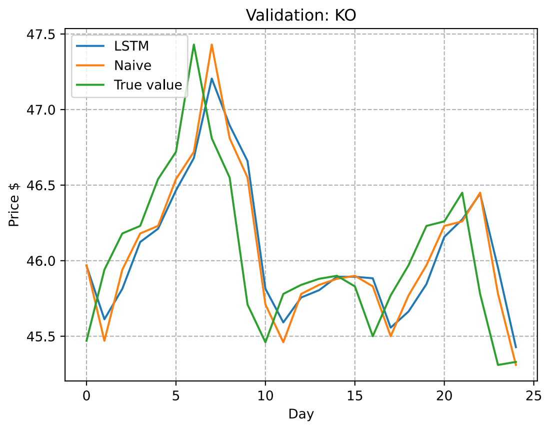
(c) NFLX (d) NVDA

Figure 34: Predicted prices vs actual prices of the stocks INTC, KO, NFLX and NVDA, generated by the vanilla LSTM with all features

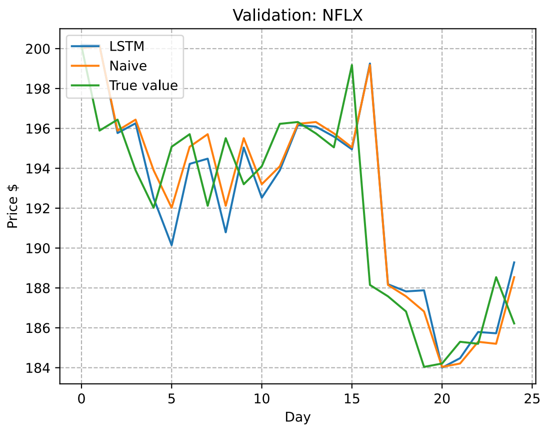
In all of the graphs, the predicted price seems to be almost the same as the true price shifted one step to the right as can be seen in Figure 35, which is generated by the vanilla LSTM model that uses all features. This is especially apparent when comparing directly with the naive model, for instance in Figure 35b or Figure 35c where the similarities between the LSTM predictions and the naive model are clear. In other words it seems that the model predicts the next price to be the current price plus some value that the results seem to suggest are noise probably caused by the additional features or the high number of weights that are being tuned during training.



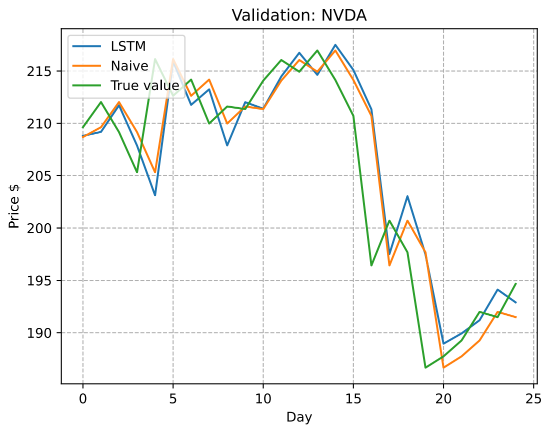
(a) INTC



(b) KO



(c) NFLX



(d) NVDA

Figure 35: Predicted prices vs actual prices of the stocks INTC, KO, NFLX and NVDA, generated by the vanilla LSTM with all features, zoomed in on the 25 first predictions

The HD stock was analysed further to gain insight and to comment on the situations where the model seems to predict accurate values. It can be seen in Figure 36 that the model at multiple time steps seemingly produces accurate predictions of the price. Some examples are at time step 2, at time step 9, at time step 13, at time step 16, at time step 18. Many of these can be explained by the model being shifted one step to the right in relation to the actual prices and that the actual price from one step to the next did not change much. But in some cases, the model accurately predicted the price even though the actual price changed noticeably from the day before. One example is at time step 28, another example, although to a lesser degree is at time step 18. This can be explained either by randomness or that the model actually has some predictive powers. The former explanation seems to be the most probable, as can be seen if the predicted graph is shifted to the left. The prediction is heavily influenced by the current price, and the seemingly accurate predictions appear to be due to coincidences. As the current price and next day prices are really close in most situations, there will be cases where the prediction and actual value will coincide. As the predictions constantly lag behind the actual value, financial gain is difficult, which is emphasized by the low DA scores.

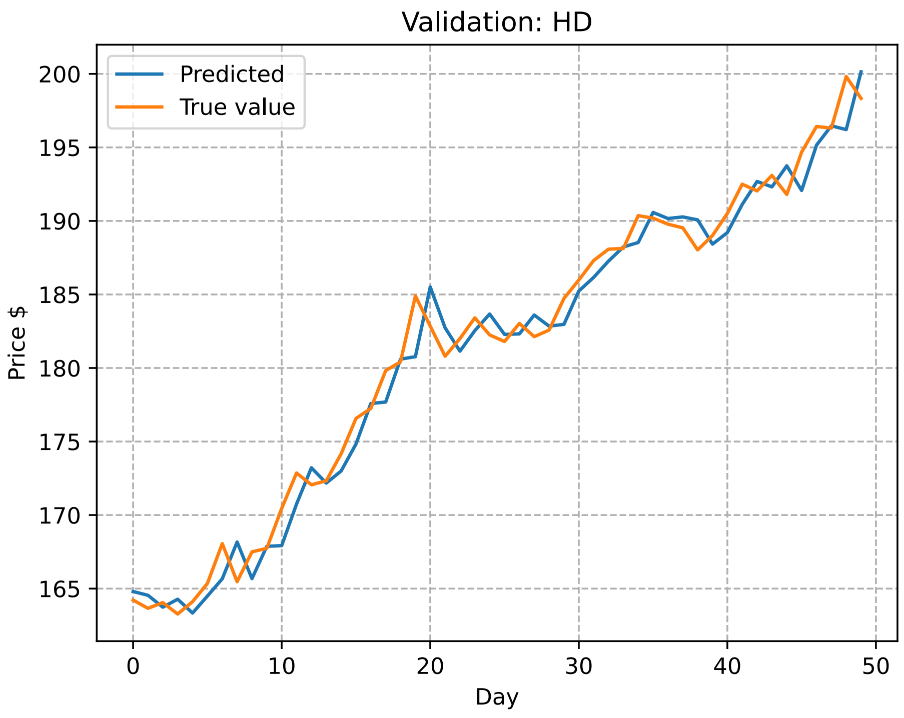


Figure 36: The vanilla LSTM model predictions on validation set for HD, zoomed in on the 50 first predictions.

Summarizing the observations

Generally, it is evident that the predicted next day price is very close to the actual price the day before, suggesting that the models are not able to make use of the additional information to offset this overdependence on the current price. This was also something that was an issue with the baseline models. However, there are some results that indicate that including some additional information could be beneficial, at least to somewhat outperform the simple 1-step-behind model, as can be seen in Table 16. The experimentation was continued with the context module in Section 8.6, to see if this observation would persist, and to examine what would happen with such a configuration.

Model	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
1-step-behind		1.342%	3.153	52.42	52.61%
Vanilla [160]	price, trendscore	1.347%	3.119	51.15	53.9%

Table 16: The best model found when predicting the next price

8.6 Introducing the context module

As the results in Section 8.5 indicated that adding additional layers only improved complexity of the model without improving the performance, experimentation with these models was not carried further.

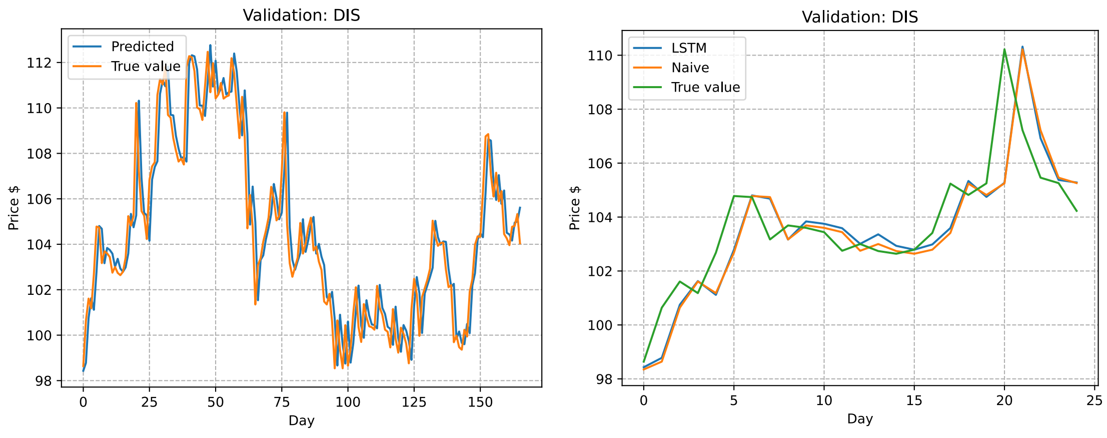
The averaged results of the experiments on the vanilla LSTM with context module are presented in Table 17. All results can be found in Table 62 in the Appendix. From the results in Table 17 and Table 62, and from associated plots (exemplified with Figure 37), it can be observed that there are several similarities with the results in Section 8.5. This makes sense, as the vanilla LSTM with context module is an extension of the vanilla LSTM without.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
1-step-behind		1.342%	3.153	52.42	52.61%
Vanilla [160]	price, trendscore	1.347% (#1)	3.129 (#1)	51.6 (#1)	52.53% (#2)
Vanilla [160]	price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.351% (#3)	3.149 (#2)	53.06 (#3)	52.65% (#1)
Vanilla [160]	price	1.349% (#2)	3.162 (#3)	52.81 (#2)	51.96% (#3)
Vanilla [160]	price, open, high, low, volume, direction, change	1.373% (#4)	3.237 (#4)	57.18 (#4)	49.98% (#5)
Vanilla [160]	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	1.42% (#5)	3.449 (#5)	62.76 (#5)	50.42% (#4)

Table 17: Predicting the next price using the context module.

First, it can bee seen that all the metrics seem correlated, meaning that the top configurations usually have a high rank in all metrics. With the three regression metrics MAPE, MAE and MSE, this is intuitive as all three metrics measure the regression performance. It is interesting that the DA score also follows this behavior, raising the question whether a better regression score consistently will result in better direction accuracy, or whether this is purely coincidental. As the results in Section 8.5 also exhibit similar tendencies, the former can not be ruled out yet. It should be noted that all DA scores hover around 50%, possibly facilitating this behavior. It is also possible that a better regression score results in a model more similar to the 1-step-behind model, causing models that achieve this to have improved performance compared to a model not able to mimic this to the same sufficient degree since this ultimately makes the model more similar to the random guessing model (see Section 8.1.1) in terms of DA. If this is the case, it would indicate that the LSTM model with context module is not able to extract useful information from the other features, at least in terms of DA, or that it is too difficult for the model to include the other features sufficiently due to the high correlation between the output and “price”.

A second point is that similar to the models in Section 8.5, the vanilla LSTM model with context module appeared to approximate the naive model, resulting in the model consistently lagging behind the actual value by one step. Figure 37 illustrates this, showing the plots of the stock DIS both on the whole validation set and zoomed in on the 25 first days, generated by the model trained on “price” and sentiment data. Examining Figure 37b closely, it can be observed how closely related the LSTM predictions are to the naive model predictions, despite including other features than “price”.



(a) DIS on the whole validation data set

(b) DIS zoomed in on the 25 first days

Figure 37: Predicted prices vs actual prices of the stock DIS, generated by the vanilla LSTM with context moduel on Price + Sentiment data

In terms of feature subsets, the results from the vanilla LSTM with context module seem to indicate that there could be some merit in using additional features. As was seen in Table 17, both the combination of “price” and trendscore data and “price” and sentiment data produced results that were overall better than only “price”. To investigate whether this would consistently hold true, runs with all of these combinations were done, each of them 10 times. The results of the best runs, respectively, are shown in Table 18, with all the results presented in Table 63 in the Appendix.

Model	Features	MAPE	MAE	MSE	DA
1-step-behind		1.342%	3.153	52.42	52.61%
Vanilla [160]	price, trendscore	1.343%	3.116	51.39	51.88%
Vanilla [160]	price	1.345%	3.125	51.93	53.17%
Vanilla [160]	price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.352%	3.138	51.58	52.2%

Table 18: Optimising model related to each feature subset 10 times. The table shows the best individual run for each feature subset.

Analysing the results in Table 18 and Table 63, it seems that including "trendscore" resulted in a better performance overall compared to using "price" as the only feature. Using only "price" resulted in a best run with better DA compared to adding "trendscore", however, when looking at the averaged scores, adding "trendscore" resulted in better performance on all metrics. It should be noted that the improvements are marginal and not enough to solve the issue with the model lagging one step behind the actual values.

Summarizing the observations

Overall, adding the context module did not seem to improve performance, with the model exhibiting similar issues and tendencies as the models experimented with in Section 8.5. This includes the fact that the model seems to approximate the simple 1-step-behind model, thus constantly predicting one day behind, but also that "trendscore" again resulted in the model that achieved the best overall performance. One point that was made clear in this part of the experiments was that improved regression metrics do not always translate to a better direction accuracy, which is also illustrated in Table 19.

The lack of sufficient improvements could be due to the underlying issues related to the experimental setup. The experimental setup was therefore modified in Section 8.8.

Model	Features	MAPE	MAE	MSE	DA
1-step-behind		1.342%	3.153	52.42	52.61%
Vanilla [160]	price, trendscore	1.343%	3.116	51.39	51.88%

Table 19: The best model found when using context

8.7 Predicting price by optimising on price and direction

Model	Features	Mean MAPE	Mean MAE	Mean MSE	Mean DA
1-step-behind		1.342%	3.153	52.42	52.61%
Vanilla [160]	price	1.355% (#1)	3.161 (#1)	53.08 (#2)	52.05% (#1)
Vanilla [160]	price, trendscore	1.361% (#2)	3.188 (#2)	52.64 (#1)	52.0% (#2)
Vanilla [160]	price, open, high, low, volume, direction, change	1.385% (#3)	3.378 (#3)	61.12 (#3)	50.61% (#4)
Vanilla [160]	price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.42% (#4)	3.538 (#4)	67.34 (#5)	50.79% (#3)
Vanilla [160]	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	1.437% (#5)	3.546 (#5)	66.67 (#4)	49.56% (#5)

Table 20: Predicting price and direction.

Before modifying the experiment to predict price change instead of the price, an experiment where the LSTM model is trained to output both the next direction and the next day price was conducted. This experiment was entirely for experimental purposes, to see whether training on direction as well would lead to increased DA scores.

Table 20 presents the results from this experiment. From these results, it is observable that this change in configuration does little to improve the performance. Analysing the results in both Table 20 and Table 64 in the Appendix, it can be argued that the changes made actually make the model worse. For instance, comparing the MSE of this model and the results from the previous sections reveals that this configuration more frequently obtains scores in the 60s. The decrease in performance is probably due to the added complexity without bringing sufficient positive contributions to compensate for this. In short, it seems to be more challenging to predict two outputs compared to one, which can be analogous to real life where learning multiple things at the same time often is more difficult compared to focusing on one thing.

Summarizing the observations

The results indicated that improvements would not be feasible with this configuration, which seemed to derive mostly from the additional complexity with prediction of multiple outputs as done in this experiment. With these evidences at hand, investigating this configuration further was decided against.

8.8 Additional experiment: Predicting price change

As the experimentation in Section 8.5 and 8.6 revealed difficulties with good performance on the initial setup, the setup was changed to accommodate these challenges. Instead of predicting the next day price, the task was altered to predict the price change.

8.8.1 Predicting price change without using context

Presented in Table 21 are the top 5 averaged results when predicting price change using the same feature subsets as in the previous experiments. Additionally, Table 22 shows the best run from selected feature subsets. Together with associated plots (examples provided in Figure 38), this produces some noticeable observations and tendencies. Results from all runs can be found in Table 65 in the Appendix.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	**3.153**	52.42	52.61%
Vanilla [160]	price, open, high, low, volume, direction, change	1.356% (#1)	**3.128 (#1)**	52.38 (#1)	51.95% (#2)
Vanilla [160]	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	1.358% (#2)	3.161 (#2)	53.64 (#2)	52.0% (#1)
Vanilla [160]	price, trendscore	1.923% (#3)	4.498 (#3)	109.6 (#3)	51.38% (#5)
Vanilla [160]	price	1.927% (#5)	4.62 (#4)	119.9 (#4)	51.41% (#4)
Vanilla [160]	price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.925% (#4)	4.725 (#5)	125.1 (#5)	51.42% (#3)

Table 21: Price change prediction without context module.

Model	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
Baseline		1.342%	**3.153**	52.42	52.61%
Vanilla [160]	price, open, high, low, volume, direction, change	1.348%	3.104	51.74	53.09%
Vanilla [160]	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	1.356%	3.125	52.89	53.01%

Table 22: The best individual runs for each selected feature subset.

One observation related to the results presented in Table 21 is that the models that use all trading features have relatively low MAEs compared to the other models. This can also be seen in Table 22, which depicts the best run for these feature subsets. Both results achieved higher scores compared to the baseline model based on the naive model, suggesting that there could be some benefits to using the additional trading data. In terms of the other metrics, this configuration does not consistently beat the baseline model, and in terms of MAPE it always performs worse than the baseline model.

The model using only price yielded relatively bad results, implying that the LSTM model has difficulties extracting information from the current price in order to predict the next change. This is especially evident when compared to the previous experiments conducted. Analysing the results further suggested that this configuration seemed to be stuck in bad local optima in all runs. To gain better understanding on how to utilize the trading data, the trading data was divided into smaller subsets in order to recognize the features that lead to the favorable results as seen in Table 21 and Table 22. The trading features were divided into groups of similar features; [“change”], [“price”, “open”, “high”, “low”], [“volume”] and [“direction”].

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Vanilla [160]	price, open, high, low, volume, direction, change	1.356%	3.128	52.38	51.95%
Vanilla [160]	**price, open, high, low, volume, direction, change**	1.374% (#1)	3.18 (#1)	53.93 (#2)	51.3% (#3)
Vanilla [160]	change	1.386% (#2)	3.189 (#2)	53.81 (#1)	49.32% (#5)
Vanilla [160]	price, open, high, low	1.957% (#4)	4.851 (#3)	135.0 (#3)	51.72% (#1)
Vanilla [160]	volume	1.951% (#3)	5.07 (#4)	156.4 (#4)	50.91% (#4)
Vanilla [160]	direction	4.316% (#5)	8.737 (#5)	293.1 (#5)	51.69% (#2)

Table 23: Predicting price change, analysing different subsets of the trading data. The results in the yellow row are extracted from Table 21.

Table 23 (all results are in Table 66 in the Appendix) shows that even with the same features, the results can differ relatively much. The mean MAEs were 3.128 and 3.180 in two different runs. This must be taken into consideration when judging the results. Since the results are not consistent, one run does not necessarily determine whether the model performs well or not. Differing results between runs related to using only trading data can be seen in Table 24.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	**3.153**	52.42	52.61%
Vanilla [160]	price, open, high, low, volume, direction, change	1.356%	**3.128**	52.38	51.95%
Vanilla [160]	price, open, high, low, volume, direction, change	1.374%	**3.18**	53.93	51.3%
Vanilla [160]	price, open, high, low, volume, direction, change	1.355%	**3.139**	52.69	51.74%

Table 24: The metric result variations using the same features; trading data

Another observation that can be made from the results in Table 23 is that using only "change" yields almost as good mean results as "change" in addition to the other trading features. Also, runs without "change" seem to perform noticeably worse, especially evident when looking at the MSE scores, as omitting "change" results in MSE in the 100s compared to MSE in the 50s. Because the model is predicting the next change, it makes sense that the current change is a more favorable feature than price. In Table 25 (the rest of the results are in Table 70 in the Appendix) the model using only price is compared against the model using only change, illustrating the described behavior.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Vanilla [160]	change	1.368% (#1)	3.176 (#1)	53.71 (#1)	49.27% (#2)
Vanilla [160]	price	1.9% (#2)	4.541 (#2)	114.7 (#2)	51.23% (#1)

Table 25: Comparing using only price to using only price change when predicting the next change.

As it is clear that “change” is a more relevant feature than price when predicting price change, the experiment was redone with “change” as the main feature instead of “price”. The generated results are presented in Table 26, with results from all the runs shown in Table i 67 in the Appendix.

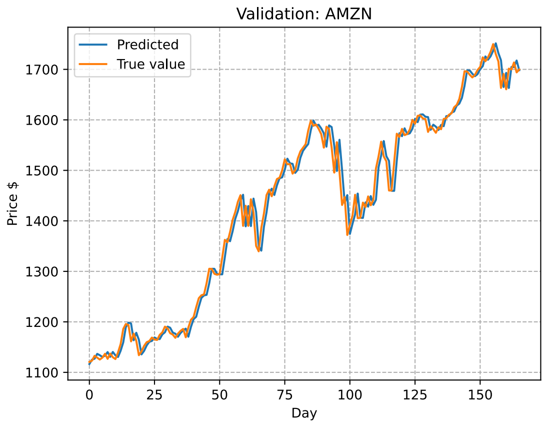
Model	Features	Mean MAPE	Mean MAE	Mean MSE	Mean DA
Baseline		1.342%	3.153	52.42	52.61%
Vanilla [160]	**price, open, high, low, volume, direction, change**	1.356%	**3.128**	52.38	51.95%
Vanilla [160]	change, open, high, low, volume, direction, price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	1.364% (#3)	**3.137 (#1)**	52.34 (#1)	51.85% (#1)
Vanilla [160]	**change, open, high, low, volume, direction, price**	1.355% (#1)	**3.139 (#2)**	52.69 (#2)	51.74% (#2)
Vanilla [160]	change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.361% (#2)	3.167 (#3)	53.52 (#3)	50.59% (#3)
Vanilla [160]	change, trendscore	1.416% (#4)	3.204 (#4)	55.21 (#5)	49.64% (#4)
Vanilla [160]	change	1.467% (#5)	3.211 (#5)	54.34 (#4)	49.44% (#5)

Table 26: Price change prediction without context module.

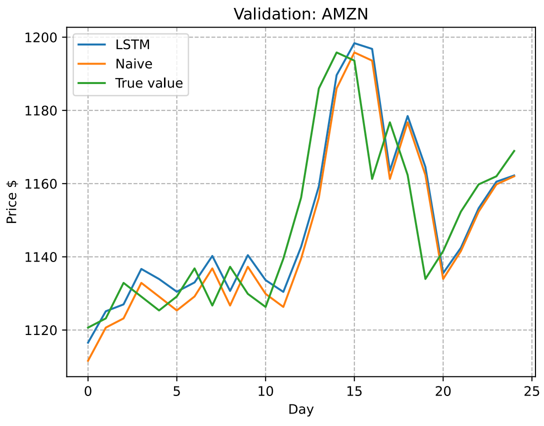
The results presented in Table 26 show that using the “change” as the main feature resulted in more stable performance in general. The mean MAPE is always in the range [1.364%, 1.467%], the MAE in [3.137, 3.211] and the MSE in [52.34, 55.21], whereas when always including price, the MAPE was in the range [1.356%, 1.927%], the MAE in range [3.128, 4.725] and the MSE in the range [52.38, 125.1]. This is an indicator that “change” is necessary in order to be able to compete with the baseline model. What is interesting is that using any combination of additional data resulted in an improvement over using “change” alone, suggesting that including this data improves the prediction capabilities. However, compared to the baseline model, none of the feature subsets seems to be able to outperform this model convincingly, suggesting that including these additional features is not enough to improve the performance sufficiently. Even when analysing the feature subset showing the most potential, trading data, further by dividing into smaller subset, the results do not deviate from this tendency. The results from this analysis are shown in Table 27, with Table 68 in the Appendix presenting more details. The results indicate that some sort of combination of all the trading features is necessary to achieve the best performance achievable with this configuration, as none of the smaller subsets matched the performance of them combined.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	**3.153**	52.42	52.61%
Vanilla [160]	change, open, high, low, price	1.434% (#3)	3.2 (#2)	54.32 (#1)	50.06% (#1)
Vanilla [160]	change, volume	1.395% (#2)	3.196 (#1)	54.36 (#3)	49.87% (#2)
Vanilla [160]	change, direction	1.389% (#1)	3.207 (#3)	54.35 (#2)	49.8% (#3)

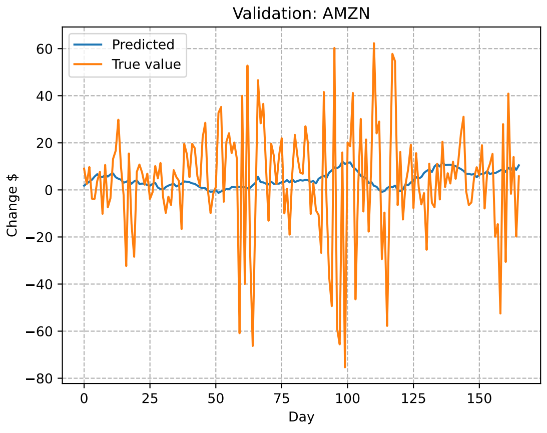
Table 27: Splitting up the trading features in order to identify patterns.



(a) AMZN results



(b) AMZN results, 25 first predictions



(c) AMZN price change predictions

Figure 38: LSTM without context module predicting price change on the stock AMZN, using all trading data

Examining the related plots highlights the underlying reasons to why the LSTM model seems to not be able to convincingly outperform the naive model. Figure 38 depicts plots related to the AMZN stock, generated by the LSTM model utilizing trading data. It should be noted that Figures 38a and 38b are from the same run, while Figure 38c is from a different run. This is due to a rerun of this particular experiment that overwrote the previous plots, making them unavailable. Nonetheless, they illustrate the overall tendencies that are present both for this stock individually and for all stocks in general. From the plots, especially when examining Figure 38c, it can be seen that the model is not able to accurately predict the actual price changes. Most predictions hover around 0 or the mean value, which also seems to be close to 0, and are rarely close to the true value. In most cases where the model predicts larger values, it seems that the predictions are wrong and do not capture the trend of that period. This translates to the model behaving close to the baseline model when adding the predicted price change to the current price, as shown in Figures 38a and 38b. This might also be the reason for the reduced DA scores when looking at averages, as the predicted price changes introduce randomness, which was shown earlier to not perform as well as the 1-step-behind model.

8.8.2 Predicting price change using context

As experimentation in Section 8.8.1 showed that "change" was necessary to produce stable and better results, experimentation with this was extended to the LSTM model with context module. This was to see whether introducing context and making the model able to specialize would mitigate the issues found in the previous section. Table 28 shows the averaged results from this part of the experiments. Table 69 in the Appendix presents all the results.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	**3.153**	52.42	52.61%
Vanilla [160]	change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.347% (#1)	**3.127 (#1)**	**52.08 (#1)**	51.81% (#1)
Vanilla [160]	change, open, high, low, volume, direction, price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	1.366% (#3)	3.153 (#2)	52.92 (#2)	51.02% (#2)
Vanilla [160]	change, open, high, low, volume, direction, price	1.364% (#2)	3.165 (#3)	53.3 (#3)	50.63% (#3)
Vanilla [160]	change	1.412% (#5)	3.17 (#4)	53.33 (#4)	50.33% (#5)
Vanilla [160]	change, trendscore	1.401% (#4)	3.184 (#5)	54.48 (#5)	50.51% (#4)

Table 28: Price change prediction with context module.

As with other experiments, the results in Table 28 seem to suggest that the regression score is correlated with the direction accuracy score, meaning that better regression scores suggested better DA scores. Likewise, the results seem to suggest that the current configuration might also exhibit the same issue as with former experiments, which was being too similar to the naive model. The results also suggest that including some additional features lead to improved performance compared to using "change" only, which is a result similar to those achieved in Section 8.8.1. Despite the improvements, only the configuration using sentiment data managed performance comparable or surpassing the performance of the baseline model. The performance is only marginally better, and it should be noted that in terms of MAPE and DA, the naive model consistently performs better. This is an indication that the model is not able to make use of the additional data properly, or that information in the available data that can be utilized to improve prediction might not exist. Another possibility is that the feature subsets might consist of too many features, with some features having positive impact while other contribute negatively. This was examined further in Section 8.8.3. However, when examining the related plots, the former theory seems to be the more likely one. Figure 39 depicts the LSTM model with context module predicting the price change of the FB stock, using trading data. This figure illustrates the general trend across all stocks and feature subsets, and shows that the LSTM with context module exhibits issues similar to the model without.

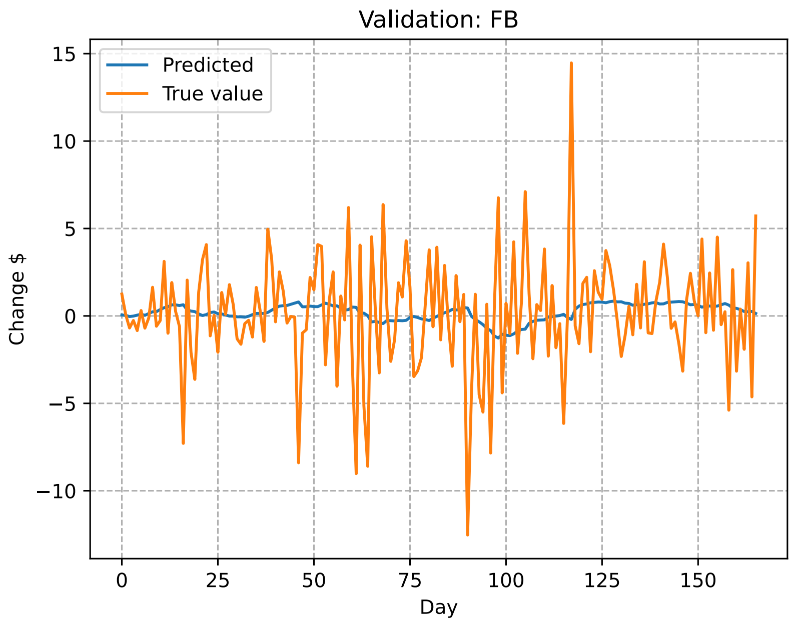


Figure 39: LSTM with context module predicting price change on the stock FB, using trading data

An interesting point that could be observed from the associated plots was that overall (examples found in Section B.6 in the Appendix), the predicted price change values tend to be positive more often than not, suggesting that the mean of the actual values seems to be slightly above 0, indicating an overall positive growth of the market over time. This holds true with real world evidence, as over time, the market has had the tendency to increase in value overall.

8.8.3 Examining additional data further

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	**3.153**	52.42	52.61%
Vanilla [160]	change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.347%	**3.127**	52.08	51.81%
Vanilla [160]	change, positive, negative, neutral	1.346% (#1)	**3.123 (#1)**	52.16 (#1)	51.33% (#2)
Vanilla [160]	change, positive_prop, negative_prop, neutral_prop	1.364% (#2)	3.145 (#2)	52.96 (#2)	51.26% (#3)
Vanilla [160]	positive, negative, neutral	1.407% (#3)	3.238 (#3)	55.2 (#4)	51.53% (#1)
Vanilla [160]	positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.418% (#4)	3.241 (#4)	55.17 (#3)	51.18% (#4)
Vanilla [160]	positive_prop, negative_prop, neutral_prop	1.486% (#5)	3.338 (#5)	60.02 (#5)	51.0% (#5)

Table 29: Identifying how much each feature subset contributes to the configuration that yielded the best results in Table 28.

In order to identify more specifically which features contribute to the improved results compared to using “change” alone, the features were divided into smaller subsets, and trained on the LSTM model with context module. The features investigated further were the sentiment data based on the results in Section 8.8.2, with “trendscore” and trading data added for experimental purposes, and were at first divided into the following subsets: [change + sentiment], [change + sentiment proportion], [sentiment], [sentiment + sentiment proportion] and [sentiment proportion]. This was primarily to see what features contributed to improved performance, and what features contributed negatively, but also to investigate whether this would consistently hold true. The results are shown in Table 29, with Table 71 in the Appendix giving a more detailed view of the results.

Table 29 shows that the combination of change and sentiment scores yielded virtually as good results as the combination of change and all sentiment related features and is the only configuration that yielded better MAE and MSE than the naive model. These results seem to imply that sentiment proportion features are less suitable when predicting the price change compared to the other features, indicating that any improvements made including sentiment data did not stem from these features. The results also suggest that examining these features further could be interesting, thus the features were again divided into even smaller feature subsets: [change + positive], [change + negative], [change + neutral], [change + positive + negative], [change + positive + neutral], [change + neutral + negative]. The produced results are presented in Table 30. All results from this experiment are shown in Table 72 in the Appendix.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	**3.153**	52.42	52.61%
Vanilla [160]	change, positive, negative	1.346% (#1)	3.126 (#1)	52.13 (#1)	51.85% (#1)
Vanilla [160]	change, positive	1.347% (#2)	3.129 (#2)	52.14 (#2)	51.34% (#3)
Vanilla [160]	change, positive, neutral	1.349% (#3)	3.131 (#3)	52.46 (#3)	51.61% (#2)
Vanilla [160]	change, negative	1.361% (#4)	3.144 (#4)	52.95 (#5)	50.92% (#4)
Vanilla [160]	change, neutral	1.398% (#6)	3.155 (#5)	52.70 (#4)	50.49% (#5)
Vanilla [160]	change, negative, neutral	1.367% (#5)	3.162 (#1)	53.33 (#6)	50.09% (#6)

Table 30: Identifying how much each feature subset contributes to the configuration that yielded the best results in Table 29.

In Table 30, it can be observed that the “positive” feature seems to be included in all the feature subsets that overall achieved the best performance, suggesting this feature to be the main reason for the improvement over other feature subsets, and the reason to an improved MAE and MSE compared to the baseline model. Although "positive" seems to give metric improvements in terms of MAE and MSE, it still does not seem to be meaningful improvements, which is emphasized by the inferior DA scores and the related plots (examples provided in Section B.6 in the Appendix) that seem to exhibit similar issues as the previous experiments when predicting price changes. Just for experimental purposes, the "positive" feature was combined with trendscore data and trading data. Results from these runs are presented in Table 31 and Table 32. Results from all runs from these experiments are present in Table 73 and Table 74, both in the Appendix. None of the runs seemed to yield any positive results or improvements, giving indication that these features might not be the answer to outperforming the baseline model and producing result applicable to real world usage.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Vanilla [160]	change, positive	1.347%	3.129	52.14	51.34%
Vanilla [160]	change, positive, trendscore	1.351%	3.134	52.34	51.45%

Table 31: Introducing trendscore.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Vanilla [160]	change, positive	1.347%	3.129	52.14	51.34%
Vanilla [160]	change, positive, volume, direction	1.349% (#1)	3.134 (#1)	52.64 (#1)	51.1% (#1)
Vanilla [160]	change, positive, open, high, low, price, volume, direction	1.381% (#2)	3.233 (#2)	55.79 (#2)	50.37% (#2)
Vanilla [160]	change, positive, open, high, low, price	1.393% (#3)	3.247 (#3)	55.84 (#3)	49.4% (#3)

Table 32: Introducing trading data

8.8.4 Experimenting with feature engineering

The previous experiments displayed that the LSTM models might have issues with utilizing the currently available features, especially when attempting to predict price changes. In an effort to advance the performance on this task, several additional features were created. As the "change" feature seemed to have had the most impact on this task, the newly created features were all related to value changes in some way. These features, previously described in Section 4.2.1, represented the change between "price" and the price related features "open", "high" and "low". Additionally, the feature "trendscore_change" was created to represent the change of search popularity between days. All results can be found in Table 75 in the Appendix, while the averaged results are presented in Table 33.

Model	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
1-step-behind		1.342%	3.153	52.42	52.61%
Vanilla [160]	change, open_close_change, high_close_change, low_close_change	1.36% (#1)	3.179 (#2)	55.85 (#3)	51.16% (#1)
Vanilla [160]	change, trendscore_change	1.375% (#2)	3.155 (#1)	52.97 (#1)	50.36% (#3)
Vanilla [160]	change, open, high, low, volume, direction, price, open_close_change, high_close_change, low_close_change	1.38% (#3)	3.186 (#3)	54.54 (#2)	50.64% (#2)

Table 33: Using price differences between open, low and high, and close price or trendscore change.

As can be seen from the results in Table 33, it is evident that the newly created features do not have a positive impact on performance. The metric results are not as high as the highest ranked configurations in previous experiments, while the plots (examples are provided in Section B.6 in the Appendix) show little to no change, suggesting that these features did not contribute with any useful information. A detail that is interesting from this experiment is that "trendscore_change" had little impact on predictive performance, indicating either that a change in popularity does not affect the price of the stock, or that representing popularity using search popularity in the way done in this project is not the correct way of doing it.

Summarizing the observations

Looking through the experiments and results in this section, there is an adequate amount of evidence demonstrating that the LSTM model has issues finding meaningful information in the available features, making the model unable to predict the next price change accurately or more accurate than the baseline model. As Table 34 illustrates, some configurations of the model did outperform the baseline model in terms of MAE and MSE, although only marginal. However, as MAPE and (especially) DA were not improved, it was clear that the LSTM model did not perform better overall, which was highlighted by the plots showing insignificant change compared to the plots produced by the naive model. This ultimately meant that the LSTM model predictions were one step behind compared to the actual value.

As the LSTM model was not able to utilize available features in the current time step to produce accurate predictions, Section 8.9 experimented with adding lagged variables to see how this would affect the performance.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	3.153	52.42	52.61%
Vanilla [160] Without context	price, open, high, low, volume, direction, change	1.356%	3.128	52.38	51.95%
Vanilla [160] With context	change, positive	1.347%	3.129	52.14	51.34%

Table 34: The selected feature sets for the models with and without context module. Extracted from Table 21 and Table 30

8.9 Additional experiment: Predicting next price change using several time steps

Lagged variables, meaning features from earlier time steps, were introduced to see whether this would give the necessary information to assist in price change prediction. The results from this experiment are presented in Table 35 (see all results in Table 76 in the Appendix). The notation of the features is changed slightly, to reflect that earlier time steps are included. “change[0-2]” means that the “change” feature from the current time step (0), the previous time step (1) and the time step before that again (2) is used.

As is observable in Table 35, introducing the lagged variables had insignificant effect on performance. Similar to previous results, this configuration managed to achieve marginally better MAE and MSE scores, but worse MAPE and MSE. Again, this suggests that the model is not available to extract useful information to increase overall performance over the baseline model. The associated plots (examples provided in Section B.7 in the Appendix) exhibit similar issues as with previous findings, showing predictions that indicate minimal or no improvements over the prior experiments.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	**3.153**	52.42	52.61%
Vanilla [160]	change[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], price[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2], trendscore[0-2]	1.353% (#2)	3.126 (#1)	52.23 (#2)	51.33% (#1)
Vanilla [160]	change[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], price[0-2]	1.354% (#3)	3.135 (#2)	51.8 (#1)	50.95% (#2)
Vanilla [160]	change[0-2], positive[0-2]	1.35% (#1)	3.142 (#3)	52.51 (#3)	49.86% (#5)
Vanilla [160]	change[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2]	1.363% (#4)	3.167 (#4)	53.49 (#4)	50.61% (#3)
Vanilla [160]	change[0-2], trendscore[0-2]	1.386% (#5)	3.185 (#5)	53.66 (#5)	49.91% (#4)

Table 35: Using several time steps

A quick experiment was conducted on the price predictions as well, with the results implying similar issues with that configuration, further strengthening the assumption of a lack of useful information in the available data.

Summarizing the observations

The best result is presented in Table 36, compared against the baseline model. The results highlight the issues that have been a common theme across all experiments, including this, which is the fact that no configuration is able to sufficiently deviate from, and outperform, the naive model.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	**3.153**	52.42	52.61%
Vanilla [160]	change[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], price[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2], trendscore[0-2]	1.353% (#2)	3.126 (#1)	52.23 (#2)	51.33% (#1)

Table 36: Best results using several time steps, extracted from Table 35

Since all previous additions and changes to the model and configurations did not yield any significant improvements, the subsequent experiments took a step back and focused on simplifying the scope, in an effort to understand the root of the issues better and to locate the source of the problem. This starts with Section 8.10, which conducts experiments on one stock at a time.

8.10 Additional experiment: Predicting price change for one stock at a time

The scope was simplified to analyze whether the issues present in earlier experiments originated from a much lower level, or emerged from the complexity introduced due to the model having to predict multiple stocks at the same time. The experimentation started out with the stock AAPL (the stock of Apple), chosen simply because it was the first on the list of stocks, with the initial results motivating extending the individual analysis to two more stocks; FB (the stock of Facebook) and HD (the stock of Home Depot). FB was chosen to represent the stock of another highly recognizable company with a stock often mentioned in different media, similar to what the AAPL stock experiences, while HD was chosen to represent a stock that does not get as much media coverage as the two other stocks. Additionally, the HD stock is traded in lower volumes compared to the two other stocks, which reflects the position of HD in the stock market.

8.10.1 Experimenting on AAPL

Results are presented in Table 37. The rest of the results are presented in Table 77 in the Appendix. The feature subsets are similar to the ones used in earlier experiments, with additional runs including lagged variables.

From the results in both Table 37 and Table 77 in the Appendix, one observation is particularly interesting. The LSTM model seems to consistently be able to outperform the baseline model in terms of DA. This is interesting since it is unprecedented in this project. The best runs, done with “change” and “trendscore” and all features with lagged variables, managed DA scores of 58.79% and 56.97%, respectively, which are significantly better than the baseline model at 48.48%. Even the averaged scores on these feature subsets managed to outperform the baseline with a noticeable margin.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.025%	1.777	5.888	48.48%
Vanilla [160]	change	1.023% (#1)	1.772 (#1)	5.843 (#1)	49.09% (#6)
Vanilla [160]	price[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], change[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2], trendscore[0-2]	1.024% (#3)	1.774 (#2)	5.876 (#3)	54.14% (#1)
Vanilla [160]	change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.024% (#2)	1.775 (#3)	5.858 (#2)	49.9% (#5)
Vanilla [160]	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	1.031% (#4)	1.787 (#4)	5.893 (#4)	51.11% (#4)
Vanilla [160]	change, trendscore	1.036% (#5)	1.796 (#5)	5.924 (#5)	53.13% (#2)
Vanilla [160]	change, open, high, low, volume, direction, price	1.044% (#6)	1.809 (#6)	6.091 (#6)	51.31% (#3)

Table 37: Feature search on the AAPL stock

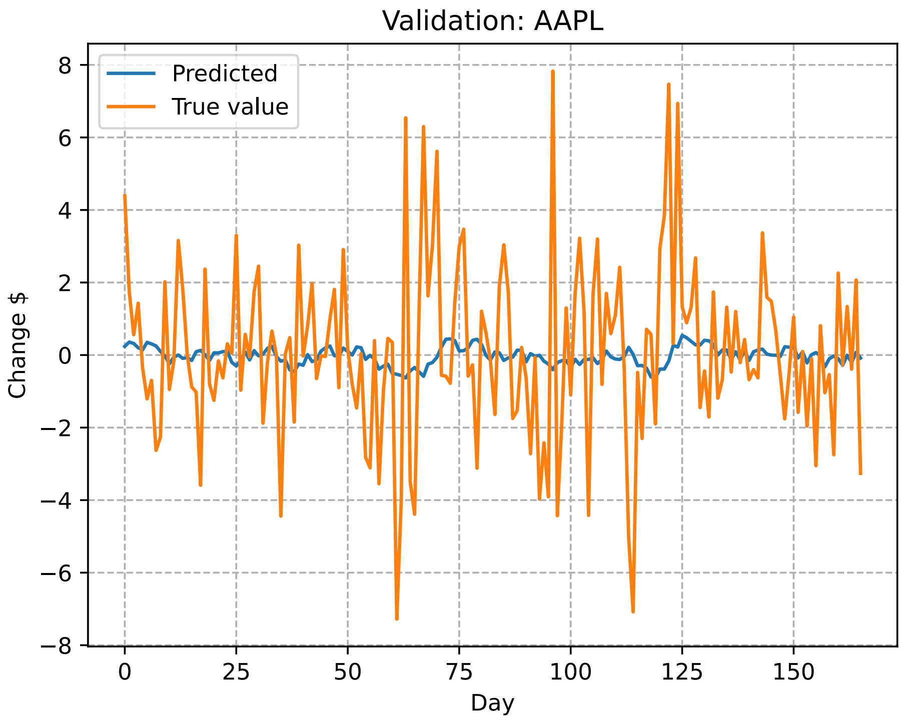
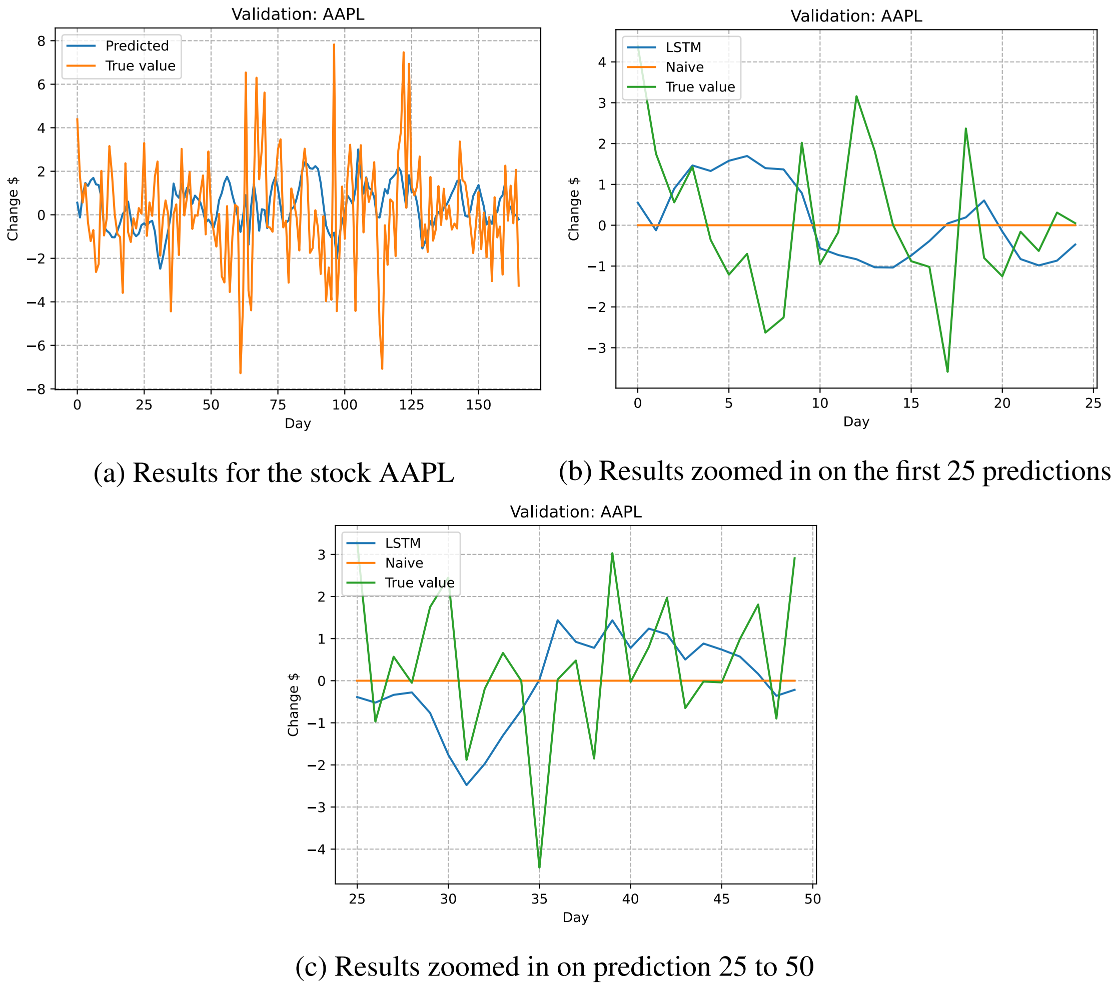


Figure 40: Predicting the next price change for the stock AAPL, using the trading data only

Even though most of the plots look similar to previous experiments, with marginal movement from the mean, as exemplified in Figure 40, there were some that exhibited movements indicating that the model was able to make use of the features for improved predictions. This is depicted in Figure 41, which are the plots related to using "change" and trendscore data. Although not as large movements compared to the true value, the plots still exhibit movements larger than previously seen. However, not all movement seems to be in the general direction of the true value. As can be seen in Figure 41b, the predictions seem to mainly move in the opposite direction of the actual values, which is particularly visible around prediction 0 to 10. At other times, the model seems to be able to follow the general trend, as illustrated in Figure 41c, where the predicted price change dips together with the true value, and later increases and hovers around a value that looks similar to the mean of that time frame (40 to 50). It should be noted that the predictions are not fully accurate, which is reflected by DA scores that still are below 60%.



(a) Results for the stock AAPL (b) Results zoomed in on the first 25 predictions

(c) Results zoomed in on prediction 25 to 50

Figure 41: Predicting the next price change for the stock AAPL, using "change" and trendscore data

The results seem to imply that the LSTM model is able to utilize the additional data to improve direction accuracy. Specifically "trendscore" seemed to have a major impact, as this feature was present in both feature subsets that managed to achieve significant improvements over using "change" alone, and even the baseline model. Additional experimentation with using the "trendscore" feature alone was conducted to examine the extent of the performance increases by this feature. The results are presented in Table 38 and in Table 77 in the Appendix, indicating that this feature had the most impact on DA scores for the AAPL stock with an average DA of 57.37% and a best score of 61.21%, compared to the 48.48% of the baseline model. The changes in regression metrics scores are much lower when comparing to the baseline results, indicating little actual movement in value from the baseline, but this goes to show how the DA and the regression metrics can be independent of each other, which has for the most part not been the case in previous experiments.

Model	Features	Mean MAPE	Mean MAE	Mean MSE	Mean DA
Baseline		1.025%	1.777	5.888	48.48%
Vanilla [160]	trendscore	1.025%	1.777	5.832	57.37%

Table 38: Using only "trendscore" on the AAPL stock

To investigate this further, and see whether this tendency was present with other stocks as well, some additional experiments were conducted, presented in the following sections.

8.10.2 Experimenting on FB

Results from experimenting with the FB stock are shown in Table 39. The results seem to again suggest that it is possible to improve prediction when introducing certain features or information. With the FB stock, "trendscore" did not seem to have large impact on performance as it did with the AAPL stock, as including it did not result in any noticeable improvements over the baseline model and using "change" alone. This seems to suggest that different features and additional data can have varied impact on prediction, meaning that not all stocks will see the same benefit in including "trendscore" for instance. What seems to have positively influenced the performance the most has been sentiment data, highlighted by the top DA result of an average of 61.82%, which improved the average by over 4% compared to having "change" only. Using the feature "change" alone lead to formidable results compared to earlier, resulting in measurements exceeding or rivaling the baseline models on all metrics. Achieving consistent results like these, as can be seen in Table 78 in the Appendix, suggests that for the FB stock, the next day price change has a stronger relationship with the current price change than experienced with other experiments in this project. Again, this suggests that different stocks have different features that can assist in prediction. Another interesting point is that adding lagged variables did not seem to improve performance, but rather decrease it noticeably, even though "change" is included. Combining "change" with certain features seemed to also decrease the score significantly, which seems to be due to either the noise added, because the features do not contribute any useful information for prediction, or due to the complexity due to the additional features added. A combination of both seems more likely when examining all the results. The results indicate that the model struggles to find good optima due to the added features that are noise, for instance "trendscore" or other trading data, just occasionally being able to move out of the less optimal as with the results using all features (found in Table 78 in the Appendix), where the model reached a DA of 56.36% on one run compared to 51.52% and 53.33% on the two others. All regression scores seem to remain stable, however.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.307%	2.328	10.62	52.12%
Vanilla [160]	change	1.288% (#1)	2.292 (#1)	10.58 (#3)	57.17% (#2)
Vanilla [160]	change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.288% (#2)	2.293 (#2)	10.57 (#2)	61.82% (#1)
Vanilla [160]	change, trendscore	1.301% (#4)	2.312 (#4)	10.6 (#4)	52.53% (#5)
Vanilla [160]	price, open, high, low, volume, direction, change	1.298% (#3)	2.31 (#3)	10.8 (#7)	53.13% (#4)
Vanilla [160]	trendscore	1.304% (#5)	2.321 (#5)	10.47 (#1)	50.3% (#7)
Vanilla [160]	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	1.307% (#6)	2.326 (#6)	10.75 (#5)	53.74% (#3)
Vanilla [160]	price[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], change[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2], trendscore[0-2]	1.317% (#7)	2.342 (#7)	10.77 (#6)	50.71% (#6)

Table 39: Feature search on the FB stock

Since most regression scores remained stable, although the DA scores improved by a significant margin, it seems like there is not a large deviation from the baseline model in terms of predicted values, implying that there is little information in the data that can be used to predict actual price change values. Examining the plots suggests the same. Figure 42 depicts the predicted price changes using “change” and sentiment data, generated by the model that achieved the best results on this configuration. This figure exemplifies how the majority of predictions looked like.

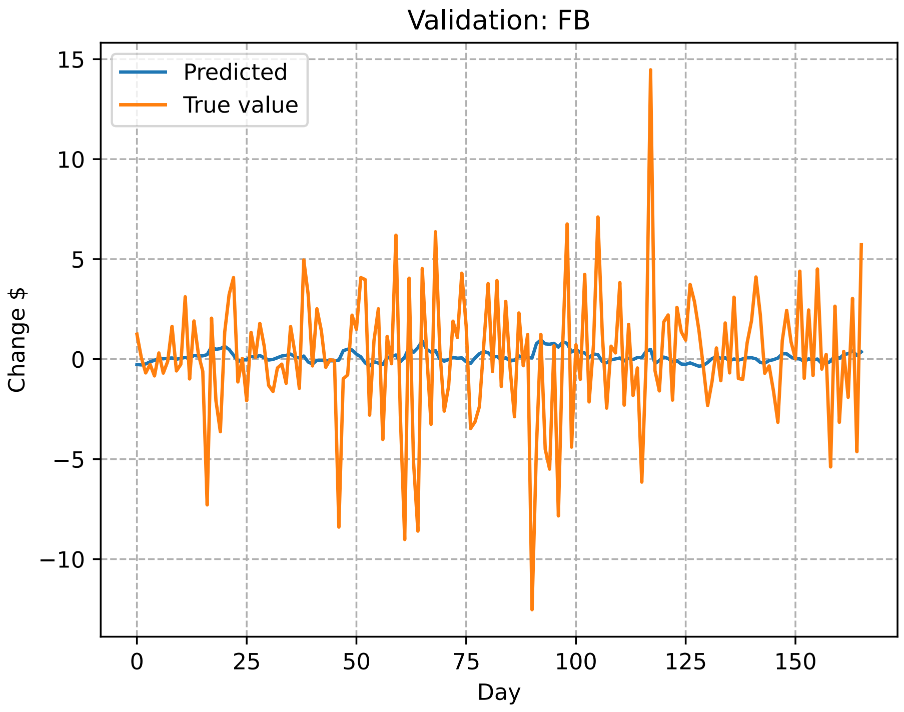


Figure 42: Predicting the next price change for the stock FB, using the “change” feature and sentiment data

Observable in Figure 42 is that the predictions are relatively close to 0, suggesting little useful information in the features, despite apparently having information to improve direction accuracy. In terms of regression performance, the LSTM model seems to therefore still not perform noticeably better than the baseline model, at least not to a degree where the regression scores can be used as a guiding tool for accurate price change predictions. It is, however, an interesting discussion whether accurate regression scores are necessary if the DA is relatively high. The discussion of this matter will be expanded on in Chapter 9.

8.10.3 Experimenting on HD

Analysis of the HD stock is present to include analysis on a stock and company that typically does not get much media coverage, especially compared to the two other stocks, AAPL and FB. The results are presented in Table 40 and in Table 79 in the Appendix.

Model	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		0.9482%	1.749	5.934	54.55%
Vanilla [160]	price, open, high, low, volume, direction, change	0.9411% (#1)	1.735 (#1)	5.924 (#2)	54.34% (#3)
Vanilla [160]	price[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], change[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2], trendscore[0-2]	0.9413% (#2)	1.735 (#2)	5.92 (#1)	55.76% (#2)
Vanilla [160]	change	0.9429% (#3)	1.738 (#3)	5.927 (#3)	53.94% (#4)
Vanilla [160]	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	0.944% (#4)	1.74 (#4)	5.972 (#4)	58.99% (#1)
Vanilla [160]	change, trendscore	0.9454% (#5)	1.743 (#5)	5.978 (#5)	50.71% (#5)
Vanilla [160]	change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	0.954% (#6)	1.761 (#6)	6.053 (#6)	49.49% (#6)
Vanilla [160]	trendscore	0.9595% (#7)	1.767 (#7)	6.082 (#7)	50.71% (#5)

Table 40: Feature search on the HD stock

Similar to the two other stocks, experimenting on the HD stock also produced results that indicated that there are possibilities for improved DA scores compared to the baseline model. In this case, the results did not indicate that combining “change” with either “trendscore” or sentiment data improved performance, but rather decreased them to a large degree, and adding the additional trade data did not seem to improve the direction accuracy. What is interesting, however, is when all the features are combined, the DA actually increases by some margin, suggesting that in some cases, the benefits are only present when features are combined, which in turn implies that there are non-linear relationships in the data that can be exploited by a model able to capture non-linear relationships. Since all three runs with this configuration achieved noticeably better DA scores than the baseline and most other configurations (see Table 40), this indicates that improvements can be made with the right feature subset. As mentioned previously, the results seem to suggest that different stocks have different relationship with the features and combination of these features.

As with most other experiments, the results imply that in terms of regression, not much improvement is made. Most predictions of price change are close to 0, regardless of the improvements in DA.

Summarizing the observations

Experimenting with one stock at a time has generated results that implied it being possible to improve prediction performance in terms of DA compared to the baseline model, and some results that indicated some additional data to be contributing to this performance increase. Although the regression metrics also saw some improvements, they were mostly marginal, and far from being accurate enough to be used as investment guidelines. The marginal improvements resulted in predictions that did not deviate far away from the baseline model predictions. There were, however, some results that depicted price change predictions considerably larger than 0, as with the AAPL stock, which had not been the case in earlier experiments. The results strengthen the assumptions that the results from the earlier experiments were partly due to the complexity of predicting multiple stocks. Additionally, the results seem to imply that each stock is affected by the features differently, only adding to the complexity of predicting multiple stocks at the same time.

Even though increases in DA were achieved in these experiments, it should still be noted that the DA scores are not considered as high or impressive in the grand scheme of things, and have yet to be tested on the test set for proper evaluation and to confirm that the reason is not due to the favorable circumstances not applicable elsewhere.

8.11 Summary and evaluation on the test set

The previous sections have only tested the LSTM models on the validation sets, in order to prevent peeking and unfairly optimizing performance on the test set. This section will present the results of selected feature subsets tested on the test set, chosen based on the observations made in the experiments. The evaluation is divided into two parts, the first part evaluating configurations analyzing all stocks, while the second part presents the evaluation on experiments running on some stocks individually. When deciding upon which model out of the three possible for each configuration to test on, the model that achieved the best results on the validation set was chosen.

8.11.1 Experiments on all stocks

Model meta	**Features**	**Mean MAPE**	**Mean MAE**	**Mean MSE**	**Mean DA**
Baseline		1.342%	3.153	52.42	52.61%
Vanilla [160] **Without** context Predicting **price** Best of 13 results	price, trendscore	1.347%	3.119	51.15	53.9%
Vanilla [160] **With** context Predicting **price** Best of 13 results	price, trendscore	1.343%	3.116	51.39	51.88%
Vanilla [160] **Without** context Predicting **price change** Mean result	price, open, high, low, volume, direction, change	1.356%	3.128	52.38	51.95%
Vanilla [160] **With** context Predicting **price change** Mean result	change, positive	1.347%	3.129	52.14	51.34%

Table 41: Most important results. Extracted from Tables 16, 18 and 34

This section starts with a brief summary of the results deemed most important, presented in Table 41, and the selected feature configurations to be evaluated, presented in Table 42 and Table 43. Only the vanilla LSTM was examined, due to the stacked LSTM showing no potential of improving performance. The corresponding test results on the validation set can be found in Section B.9 in the Appendix. To generate the results in this section, LSTM models were trained from the beginning again, as not all configurations had already trained models that could be tested on the test set. The process of training and prediction in this part is similar to the process of the experiments conducted, and described in Chapter 6 and Chapter 7.

Features
price
price, trendscore
price, open, high, low, volume, direction, change
price, positive

Table 42: Selected feature subsets when predicting price.

Features
change
change, trendscore
price, open, high, low, volume, direction, change
change, positive

Table 43: Selected feature subsets when predicting price change.

The results using the LSTM model without the context module, predicting the next price, are presented in Table 44. Results from the corresponding model predicting price change instead are presented in Table 45. Similar models with the context module were also tested, one predicting price and one predicting price change, as presented in Table 46 and Table 47, respectively.

Model	**Run**	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
Baseline			1.657%	4.428	143.2	51.47%
Vanilla [160]	1	price, trendscore	1.695%	4.578	152.1	51.72%
Vanilla [160]	0	price, positive	1.691%	4.627	149.5	51.64%
Vanilla [160]	1	price	1.669%	4.469	143.5	51.52%
Vanilla [160]	0	price, open, high, low, volume, direction, change	1.679%	4.584	148.9	53.98%

Table 44: Predicting **next price** without context on the **test** set. See all results in Table 81

Model	**Run**	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
Baseline			1.657%	4.428	143.2	51.47%
Vanilla [160]	1	change	1.666%	4.447	145.7	50.79%
Vanilla [160]	2	change, positive	1.661%	4.438	143.7	51.68%
Vanilla [160]	2	change, trendscore	1.667%	4.441	144.5	50.26%
Vanilla [160]	2	price, open, high, low, volume, direction, change	1.706%	4.519	151.2	51.84%

Table 45: Predicting **next price change** without context on the **test** set. See all results in Table 83

Model	**Run**	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
Baseline			1.657%	4.428	143.2	51.47%
Vanilla [160]	2	price, trendscore	1.664%	4.457	142.4	52.77%
Vanilla [160]	2	price, positive	1.668%	4.461	142.5	50.79%
Vanilla [160]	2	price	1.662%	4.451	143.3	50.18%
Vanilla [160]	0	price, open, high, low, volume, direction, change	1.694%	4.621	156.9	52.53%

Table 46: Predicting **next price** using context on the **test** set. See all results in Table 85

Model	**Run**	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
Baseline			1.657%	4.428	143.2	51.47%
Vanilla [160]	2	change, positive	1.668%	4.452	144.0	50.3%
Vanilla [160]	2	change, trendscore	1.672%	4.465	145.0	50.71%
Vanilla [160]	0	change	1.668%	4.458	145.7	50.22%
Vanilla [160]	1	price, open, high, low, volume, direction, change	1.68%	4.504	149.0	50.14%

Table 47: Predicting **next price change** using context on the **test** set. See all results in Table 87

Overall, the performance is comparable to the baseline model, if not slightly worse, which was not unexpected based on the performance on the validation set. There are some cases where a LSTM configuration is able to marginally beat the baseline model in terms of MSE and DA, but none are able to achieve the same or outperform the naive model in terms of MAPE. Additionally, only one model was able to achieve better MSE and DA at the same time (the LSTM with context module predicting next price with features "price" and "trendscore"), resulting in the baseline model outperforming the LSTM model in most metrics in most cases. As the results were similar to the results on the validation set, the same points of discussion are still viable with these results. These points will be deliberated further in Chapter 9.

8.11.2 Experiments on individual stocks

The results deemed as the most important from the experiments with stocks individually are presented in Table 48. The results are from the run that achieved the best performance on the validation set. All results are from experimenting with predicting price change. As each stock had different responses to different feature subsets, each stock had different feature subsets that were tested. Each stock was tested on two different feature subsets, also shown in Table 48. Since the part of the experiments analyzing stocks individually only investigated using a model predicting price change, only configuration on price change was tested upon. All results on the test set can be found in Section B.9 in the Appendix.

Model meta	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
Baseline AAPL		1.025%	1.777	5.888	48.48%
Vanilla [160] On AAPL Run 0	price[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], change[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2], trendscore[0-2]	1.014%	1.756	5.808	56.97%
Vanilla[160] On AAPL Run 1	trendscore	1.005%	1.743	5.743	61.21%
Baseline FB		1.307%	2.328	10.62	52.12%
Vanilla[160] On FB Run 1	change	1.287%	2.292	10.58	57.58%
Vanilla[160] On FB Run 1	change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.287%	2.29	10.56	64.24%
Baseline HD		0.9482%	1.749	5.934	54.55%

Vanilla[160] On HD Run 1	price[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], change[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2], trendscore[0-2]	0.9384%	1.73	5.882	57.58%
Vanilla[160] On HD Run 0	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	0.9423%	1.737	5.951	59.39%

Table 48: Most important results related to predicting prices for individual stocks. Extracted from Tables 77, 78 and 79

Table 49 and Table 88 in the Appendix show the results on the AAPL stock. While the LSTM model achieved results that were better in terms of all metrics compared to the baseline model, the same model achieved significantly worse performance when tested on the test set. The most evident difference is the decrease in DA scores for the configuration with lagged variables on all features. That specific configuration achieved a best DA score that was over 8% higher than the baseline, while having the same metric 16% lower on the actual test set. The baseline model outperformed this configuration on all metrics as well. The configuration using “trendscore” also achieved meager results compared to what the results on the validation set were indicative of. These results suggest that an improvement on one specific time period will not necessarily translate to an improvement on another time period.

Model	Run	Features	MAPE	MAE	MSE	DA
Baseline			1.486%	2.798	15.46	56.36%
Vanilla [160]	0	price[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], change[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2], trendscore[0-2]	1.491%	2.813	15.56	42.42%
Vanilla [160]	1	trendscore	1.488%	2.798	15.76	54.55%

Table 49: Predicting **next price change** on the **test** set for AAPL stock

Results from testing on the FB stock are depicted in Table 50 and Table 89 in the Appendix, showing another case of performance that does not compare to what the results on the validation set suggested. The model using "change" and sentiment data as features did, however, achieve results that managed to outperform the baseline model on all metrics, just by a lower margin compared to testing on the validation set. Again, this suggests that performance on one time period does not automatically lead to similar performance on another time period.

Model	Run	Features	MAPE	MAE	MSE	DA
Baseline			1.705%	2.675	21.26	48.48%
Vanilla [160]	1	change	1.709%	2.684	21.29	52.73%
Vanilla [160]	1	change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop	1.697%	2.664	21.05	52.73%

Table 50: Predicting **next price change** on the **test** set for FB stock

The results on the HD stock, presented in Table 51 and Table 90 in the Appendix, seem to suggest the same points that could be observed from the results of the two previous stocks. The HD stock seemed to produce results similar to the FB stock, as results on both stocks did not show nearly as good results as the validation set implied, but still managed to outperform the baseline model marginally on most metrics (with the model using all features).

Model	**Run**	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
Baseline			0.977%	1.802	6.176	47.88%
Vanilla [160]	1	price[0-2], open[0-2], high[0-2], low[0-2], volume[0-2], direction[0-2], change[0-2], positive[0-2], negative[0-2], neutral[0-2], positive_prop[0-2], negative_prop[0-2], neutral_prop[0-2], trendscore[0-2]	0.9873%	1.822	6.320	48.48%
Vanilla [160]	0	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	0.9761%	1.801	6.234	49.69%

Table 51: Predicting **next price change** on the **test** set for HD stock

Overall, the predictions did not seem to deviate from the baseline model predictions significantly, even less than when predicting on the validation set. However, some evidence suggest that marginal improvements can still be made, as can be seen in both the FB results and the HD results. As for the reasons to why the performance differences were so noticeable, the following two seem possible: the first being that features have different effect on different time periods, meaning that a feature being effective on one time period is not as effective on a different time period, the second being that the impressive performance on the validation set derived from the fact that the models are basing the early stopping on testing on the validation set, resulting in the performance being more optimized on the validation set, thus making it not applicable on the test set. The discussion of these points will be elaborated further in Chapter 9.

9 Discussion

This chapter discusses the findings and analysis done in Section 8, highlighting and elaborating on these further. Additionally, the work and results are discussed in relation to the research questions presented in Section 1. The research questions that influence the discussion in this chapter, as well as the project as a whole, are:

I) **Are there some patterns in the trading data, sentiment data and search popularity data gathered in this project that can facilitate price prediction?**

II) **Will a model based on LSTMs outperform the baseline models in predicting the next day prices of stocks?**

III) **Will introducing a context module improve prediction?**

IV) **Can one of the models with configurable parameters in this project outperform the baseline models in predicting stock prices, using the same set of parameters for every stock in this project?**

9.1 Using different data to improve prediction

To answer research question I), experimental investigations to analyze how different input features contributed to the prediction performance were conducted. The results provided insights and observations that not only answer research question I), but also provide points for further investigation.

Starting with the initial data analysis, it was evident that only those features related to price, such as "open", "high" and "low", exhibited a linear relationship with the price. A linear model, exemplified by the baseline models, is unable to capture any non-linear relationships that might exist in the data. This became evident when experimenting with the baseline models, specifically the linear regression model and ridge regression model. The difference between the simple 1-step-behind (naive) model and these aforementioned models is that they take features other than price into consideration when building a model. Results have shown that the 1-step-behind model performs better compared to the other baseline models in terms of the evaluation metrics, suggesting that models that can not capture non-linear relationships in the data are not suitable for this task. This is due to these models being forced to take into consideration features that essentially are regarded as noise for them, leading to worse performance. The initial results from the baseline models indicate that there are little or no linear relationships in the trading data (except for price related data), sentiment data and search popularity data utilized in this project that can aid in price prediction. Experimenting with the different values for the α parameter in the ridge regression model also suggested this, where a larger value for α lead to worse performance, implying that the model does not gain from increasing the weighting of the other features.

It should be noted that the ridge regression results discussed above are with the same value for α across all stocks. Further experimenting implied that having an individual α for each stock could improve performance compared to the 1-step-behind model. With the limited experimenting in this part of the project, there are no guarantees that this will hold for all stocks, or across all timelines, and could be due to optimizing on the data tested upon. A simple check suggested this assumption to hold true. When applying the same α value that achieved optimized performance for the ridge regression model in Table 11 on the another time frame from the AAPL stock (using the first 80% of the data as training, the following 10% for testing, and leaving the rest out), the performance was worse compared to the simple 1-step-behind model. This indicates that the performance increases that models such as linear regression and ridge regression achieve by using additional data are only circumstantial, and not the general truth.

Which leads to the question of whether there are non-linear relationships in the data that can be recognized and utilized to assist in prediction performance. LSTM models are example of models that can capture such non-linear relationships, which is why several configurations have been experimented with in this project. Looking at how these have performed during the experiments reveal some interesting points of discussion.

The results from when predicting the next day price showed an overwhelming indication of the model not being improved with the included data. Results during experiments suggested that some improvements with regards to MAE, MSE and DA could be made; however, these were marginal and are most likely insignificant in the grand scheme of things. The naive model seemed to consistently outperform the LSTM models in terms of MAPE, and did overall perform better compared to most feature subset configurations. There were some additional data features that in various cases indicated improvements over using “price” alone, such as the configuration including “trendscore”.

The results where “trendscore” showed potential in improving performance was an interesting case, as experimentation earlier with the ridge regression model suggested that the feature negatively impacted performance rather than improving it. These results seemed to suggest that the LSTM model was able to capture a weak link between “trendscore” to the next day price, when evaluating on the validation set. However, applying the same configuration on the test set did not produce results that indicated entirely the same. With the LSTM model without context module, results implied that including “trendscore” resulted in noticeable decrease in performance in terms of regression metrics, with a marginal improved direction accuracy compared to using “price” alone, resulting in an overall worse performance. What seems to be the case is that the models arrived on a set of weights that worked well with using “trendscore” on the validation set, that were not applicable or not as effective on the test set. Similar occurrences were present with the evaluation on the individual stocks, which showed in several cases that additional features were less effective or negatively impacted performance on the test set even though they increased performance on the validation set. Continuation of this point of discussion is further below.

The same can be said of combining “price” with other features, such as “positive” and trading data, where these configurations resulted in worse regression metrics. Marginal improvements did occur with regards to DA using “positive”, while a more significant improvement was achieved using trading data, suggesting that additional data can give small improvements in direction accuracy when included. When taking the validation set results into consideration, however, things become less clear, as the same feature subsets that improve DA scores do not consistently exhibit this behavior. Overall, including additional data seemed to not assist in better regression scores, but rather decrease performance. This is likely due to the additional data containing little information that can aid in this task, instead introducing more noise and complexity, ultimately leading to worse performance. With regards to DA, adding additional data does show some improvements, but not consistently across time frames, suggesting that this increase is not applicable in every situation of every time frame. This is not desirable, at least not when the signs of when this works and does not work are unclear. Additionally, the improvements are insignificant in the grand scheme, and would probably not have yielded any financial gains over time.

The LSTM model with context module produced results suggesting tendencies similar to the LSTM model without context module. Generally, the LSTM with context showed marginal improvements using specific additional data, such as trend data, when evaluating on the validation set compared to not using it. The same improvements were not reflected on the evaluation on the test set to the same degree, with “price” alone achieving the best performance in terms of MAPE and MAE. The LSTM with context module did seem to achieve better performance with regards to MSE and DA, both when adding “trendscore” and when adding “positive”, showing that limited improvements can be made. These improvements are so small, however, that it will make little difference from a practical standpoint. Combined with the inconsistencies with achieving better results with the additional data, the LSTM model with context module predicting price did not produce results that convincingly showed that additional data could improve performance, similar to the model without context module. Plots seemed to also support this, as the majority of plots generated deviated little from the plots generated when having “price” as the only feature (examples of plots can be found in Section C.1 in the Appendix).

Changing the configuration to predicting the next day price change instead of price seemed to reveal further insights on trying to improve prediction with the additional data. Evaluating on the validation set suggested small improvements could be made with additional trading data on the model without context, and with sentiment data on the model with context module. Examining the results on the test set did again show how inconsistent the improvements could be. While the results with the model without context on the validation set using trading data had the overall highest score, the same configuration actually achieved a noticeable worse performance on the test set of the selected feature subsets in terms of regression metrics. Using "positive" on the test set with both with and without context module did seem to give small improvements, but again, this was not enough to convincingly conclude that adding this feature will help in predicting the next day price change. The small improvements and changes to the predictions indicate that the models are not able to properly utilize information in the available data to make accurate predictions. Most predictions were values close to 0, illustrating the challenges the models faced translating the data into price change predictions. What seems to be the reason for these results is the combination of weak signals in the available data together with the complexity of the configuration. Since multiple stocks are being predicted at the same time, it is difficult to find weights that work well on all stocks. This is illustrated by the individual results that are inconsistent. Table 52 depicts on example of this, where the results on the AMD stock showed widely different scores, especially noticeable with the DA with almost a 5% difference. Even though the configurations are exactly the same, the results differ by a noticeable margin, indicating the model's incapability of achieving stable results when predicting all of the stocks at the same time.

Model	**Run**	**Features**	**MAPE**	**MAE**	**MSE**	**DA**
Vanilla [160]	0	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	2.258%	0.273	0.130	50.30%
Vanilla [160]	1	price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore	2.471%	0.301	0.148	45.45%

Table 52: Inconsistent results on the AMD stock

Moving to experimenting with one stock at a time gave some indication of how the complexity affected the results negatively, and revealed one additional issue with processing all stocks simultaneously. When processing one stock only, results seemed to be more stable, and improvements were experienced on the stock analyzed, at least during evaluation on the validation set. What seemed to be the case was that each stock was affected differently by different features. For instance, the AAPL stock seemed to react positively to “trendscore”, while the FB stock experienced the opposite since the performance decreased compared to using “change” as the only feature. Improving performance seems therefore to require that each stock is examined separately, as the available data do not have information that work across all stocks. A theory to what the causes this is that since the additional data used are mostly data related to public opinion (sentiment and trend/popularity), and stocks have different reactions to what the public opinion is on the particular stock or the particular company. Take AAPL as an example, that often experiences scrutiny and skepticism among the public, somewhat due to the size and market control the company inherits. Investors might not react as strongly to negative sentiments on this stock, as negative sentiment might be business as usual and that it has been observed over time that the company has an extraordinary loyal user base that would not change over to the competition easily. Additionally, the data only show some simple quantitative measures, such as with sentiment where only the amount of each category is registered, and do not contain details on what kind of events causes the different sentiment amounts. A more detailed categorization of sentiments, as done in the works of Bollen, Mao & Zeng (2011) (who categorized into different moods such as calm, alert, kind and happy), might therefore prove more useful on this task.

The results did, however, change during evaluation on the test set. These results revealed one more possible reason to why improving prediction using additional data is so challenging. Even though results on the validation set seemed promising, results on the test set indicated that such promising results do not necessarily carry over and are applicable across all time frames. With the AAPL stock, for instance, “trendscore” showed potential in improving direction accuracy of the predictions on the validation data, where the feature alone could result in over 8% increase in DA compared to only using “change”. Combining “trendscore” and “change”, however, decreased the DA with 4% compared to using “trendscore” only, suggesting “change” to not be a positive contributor. The configuration using lagged variables did also display improved performance over using “change” alone. This configuration managed to outperform the 1-step-behind model, which only had information on the current price for prediction, on all metrics, while the configuration using only “trendscore” showed significant improvements with regards to DA. However, none of the models managed to outperform the 1-step-behind model on any metrics during prediction on the test set, with the configuration utilizing lagged variables achieving a DA score of almost 14% below the 1-step-behind model. It should be mentioned that the best model using “change” as the only feature only managed 50.90% DA (a result close to random guessing), meaning that “trendscore” still did outperform this configuration in terms of DA. All regression metrics were similar or differed by a small enough margin to be considered insignificant. However, the configuration using lagged variables still performed much worse with regards to DA, illustrating the challenges with prediction across different time frames. Reasons for this might be that different time frames can have different factors that are effective in performance improvements. One time frame can have next day prices that are heavily correlated with the previous prices and trend information, for instance, while another might not exhibit the same properties. This is also backed up by the non-stationary behavior which seems to be the case with stock prices, based on the initial data analysis from Section 4.3, indicating, among other things, that the mean of one time frame is different than the mean of another.

Another factor causing the less promising results on the test set can be the model deciding early stopping point based on the validation set, meaning that the results on the validation set are more optimized. The weights used to predict on the validation set are not as effective on the test set, which might be due to the issues with different time frames having different properties, as discussed in the previous paragraph, but can also be due to overfitting on the validation set, meaning that weights are tuned to be overly effective on the validation time frame while worse at other time frames. This will be further elaborated in Section 9.5.2.

To summarize, it seems that the general models predicting all stocks at the same time, both predicting price and price change, do not benefit much from the additional data. This seems to be due to the complexity such a configuration adds, and that different stocks react differently to the features. With the configuration analyzing each stock individually, additional data seem to in some cases improve prediction, at least in terms of DA, noticeably. The improvements are, however, not guaranteed to be as effective, or effective at all, on other time frames. It should be noted that the performance, improved or not, is far from being appropriate for practical usage as of yet.

That the additional data do not yield any significant improvements compared to using “price” or “change” only does not necessarily mean that none of the gathered data are helpful at all in terms of prediction. Section 9.2 mainly discusses research question II), but contains elements and insights related to research question I), such as comparisons against a random model and a model consistently lagging one step behind.

9.2 Comparing of the LSTM models to the baseline models

Research question II) was introduced to give insight into how well the LSTM models performed compared to less complex models. In order to answer this question, the focus was on the test set as there was no peeking done on this set. It is therefore as much as practically possible stripped of any bias, whereas the validation set results could have been affected by the writers' own biases as well as the bias related to early stopping. As the naive 1-step behind model generally was superior to the other baseline models, the LSTMs will be compared to this model.

Related to the general model, optimising on all stocks simultaneously, there can be seen some improvements over the naive model in the metrics MSE and DA on the test set. The best MSE observed on the test set was 142.4 while the MSE related to the naive model was 143.2. The best DA observed was 53.98% in comparison to the baseline DA of 51.3%. In order to examine the statistical significance of the results, the random model was run 1000 times. Each next price prediction was the current price with a random value in the range [-0.005, 0.005] which was added before normalization in order to scale the prediction to fit the individual stocks and make the results comparable in the other metrics than DA as well. In order to decide whether a model is statistically significant, the results are compared to the top percentile of random models that yielded better results.

The DA results related to the random model, seen in Table 91 (Page 227) shows that the best DA found was 52.96%. This means that the best DA in relation to the LSTMs is statistically significant, as it has less than 0.1% chance of happening by chance if predictions have a 50% chance of being positive and 50% chance of being negative. Other DAs that were found to be statistically significant were related to predicting next price using context, seen in Table 46 (Page 128), where price and trendscore yielded 52.77%, which is in the top 0.3% percentile of the random models and where all current trading data yielded 52.53%, in the top 0.6% percentile.

The MSE results related to the random model, seen in Table 92 (Page 228) tell a different story where 10.8% of the results are superior to the best MSE found by the LSTMs on the test set. This shows that the best MSE results that the LSTM-models produced are not statistically significant.

Related to the model that optimizes on individual stocks, some observations are relevant. Related to the Facebook stock, the LSTM reached better metric scores in all metrics in relation to the naive model. 21.3% of the random models had a superior DA to the best LSTM model, as seen in Table 93 (Page 230) rendering the improvement statistically insignificant. 11.6% of the random models had a better MAPE than the best related to LSTM, seen in Table 96 (Page 233), 11.3% in relation to MAE, seen in Table 95 (Page 232) and 10.5% in relation to MSE, seen in Table 94 (Page 231). None of the LSTM results in relation to the Facebook stock were statistically significant. The same analysis was done on the Home Depot stock, leading to an equivalent conclusion.

As the only result that was found to be statistically significant is related to the DA of the general model; the focus should therefore be on this result. One interesting observation is that the same LSTM yielded worse results on the validation data set, 51.03%, seen in Table 80 (Page 210) compared to the baseline DA of 51.47%. This could be an indicator that the random model was not a good reference model to measure statistical significance. This could be because the random model decided direction with a probability of 50% while the true price moves upwards 51.47% of the time. A random model where the prediction has a 51.47% chance of predicting the next price to be higher than the current and 48.53% chance of predicting lower could therefore be a better reference model when measuring statistical significance. Seen in Table 97 (Page 234), the results show that the best result of 1000 runs for this random model is 53.37%, a little higher than the worst of the three runs of the LSTM configuration, 53.09%. This random model did not generally yield much better results than the other, and the other DAs that were found to be significant in relation to the first model were also found to be significant when compared to this random model. This is another indication that reaching 53.98% is highly unlikely to happen by chance. The evidence points to another reason for the relatively poor results on the validation set; that the model learned something significant that was not expressed on the validation set that emerged only on the test set. One evidence that points to this hypothesis is that the variance of the test set is generally closer to the training set than the variance of the validation set is. Four stocks have a validation set variance closer to the training set variance compared to the test set and training set, while 11 stocks have a test set variance closer to the training set variance compared to the validation set and training set. The variance is shown in Table 53. It seems that the stocks generally behave more similar when comparing the training set to the test set than when comparing the training set to the validation set.

Stock	**Training set**	**Validation set**	**Test set**
AAPL	728.4	69.89	**649.9**
AMD	13.69	3.076	**19.16**
AMZN	6.111e+04	**3.68e+04**	2.341e+04
BIDU	1.738e+03	188.9	**990.6**
DIS	413.8	**14.48**	9.469
FB	1.866e+03	105.8	**416.3**
HD	945.5	100.9	**166.2**
INTC	30.63	**16.19**	5.25
KO	6.106	2.563	**2.609**
NFLX	2.244e+03	4.485e+03	**1.579e+03**
NVDA	2.213e+03	425.9	**2.9e+03**
PFE	9.357	0.5578	**4.19**
QCOM	78.01	27.76	**49.58**

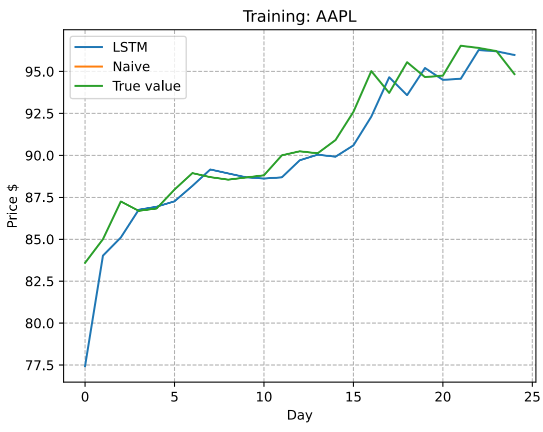
TSLA	7.625e+03	568.5	**787.9**
WMT	36.19	**52.46**	18.4

Table 53: Variance of the different stock in the different data partitions. Bold text signifies that the variance is the closest one to the training set variance for the same stock.

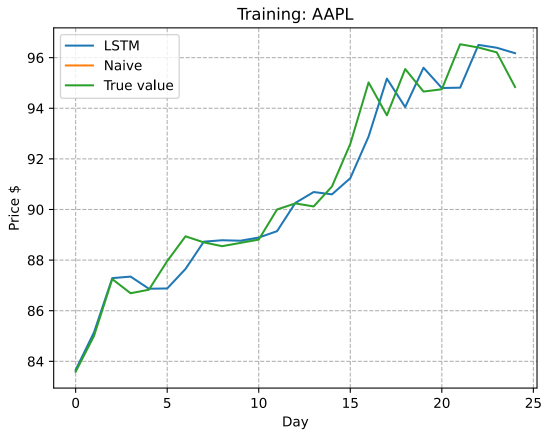
From this, the evidence points to the conclusion that there is an LSTM configuration that is superior to the naive model in terms of DA while there is no evidence that points to any of the LSTM configurations being significantly superior in the other metrics.

9.3 Including context module

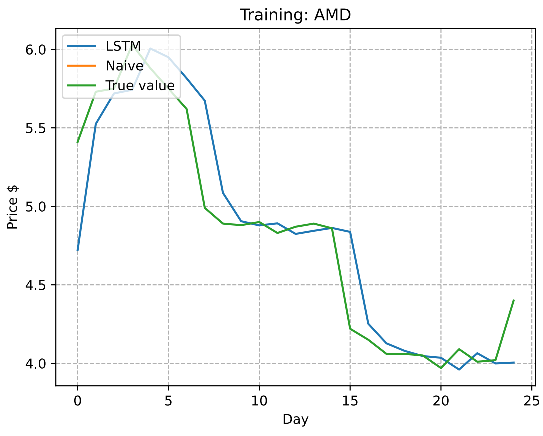
From Section 9.2 it was found that both the LSTM with and without context, predicting next price yielded DAs that were significantly better than the naive model, seen in Table 44 (Page 127) and Table 46 (Page 128). The context model that was found to generally be the best one was the model using price and trendscore, predicting next price. There were no other models that were found to be significantly better than this one in terms of MAPE, MAE or MSE. This could be seen as an indicator that including the context module improves the model. A counter argument is that none of the regression metrics related to this model were found to be significantly better than the naive model in Section 9.2. One possible explanation is that the data do not contain information that can be used to create an LSTM based model that is significantly better than the naive model in these metrics. This explanation is very hard to falsify as the number of possible LSTM configurations is very large. Another explanation could be that there are some predictive patterns in the data, but the context model in itself does not contribute significantly in order to express these patterns.



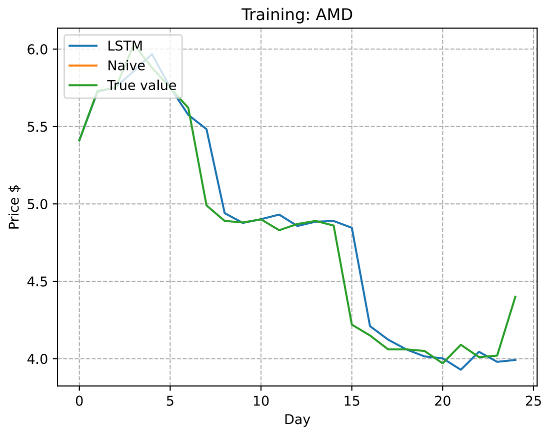
(a) Results for the AAPL stock



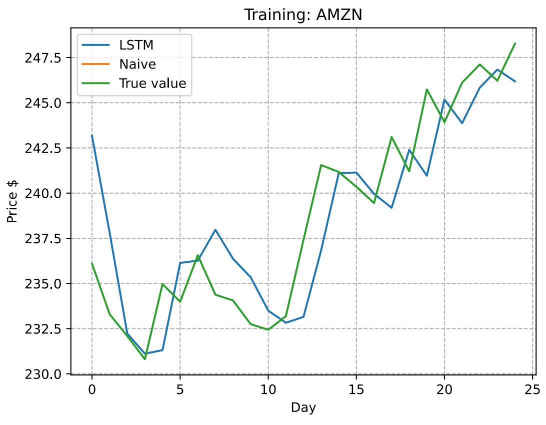
(b) Results for the AAPL stock with context



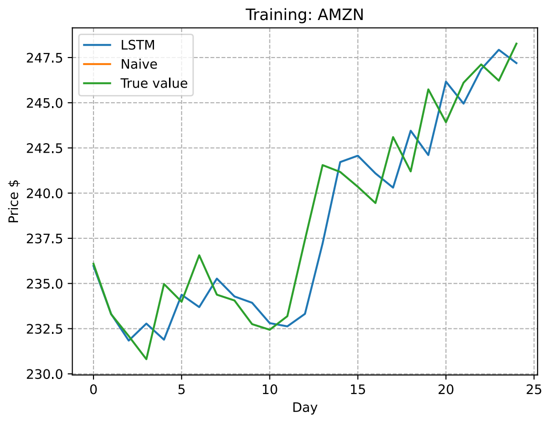
(c) Results for the AMD stock



(d) Results for the AMD stock with context

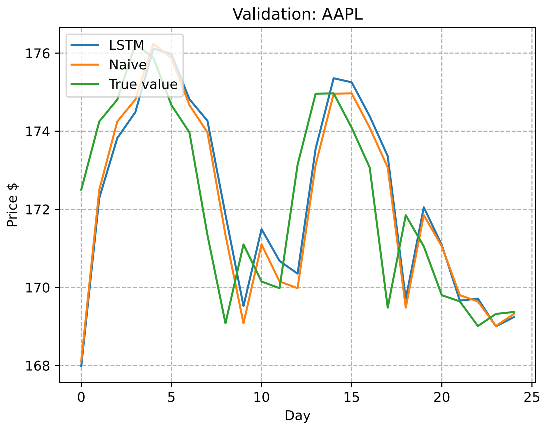


(e) Results for the AMZN stock

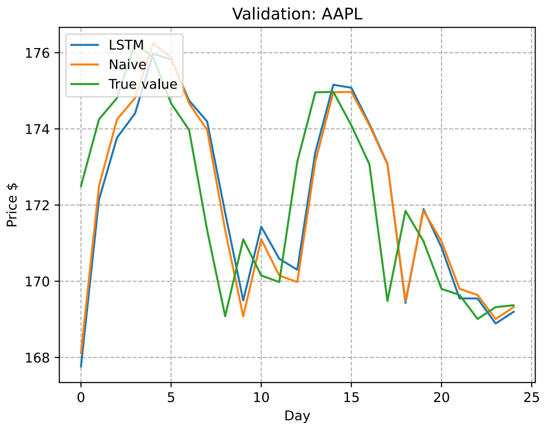


(f) Results for the AMZN stock with context

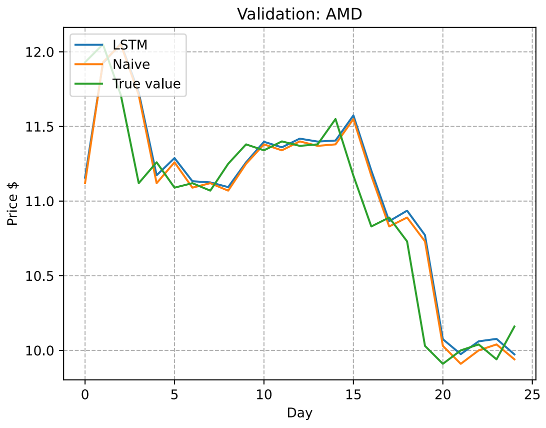
Figure 43: The first 25 time steps on the training set



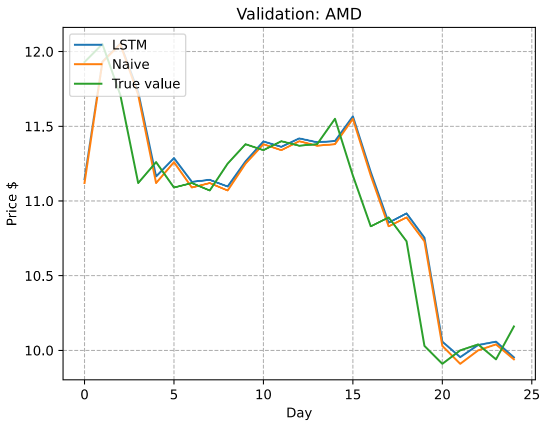
(a) Results for the AAPL stock



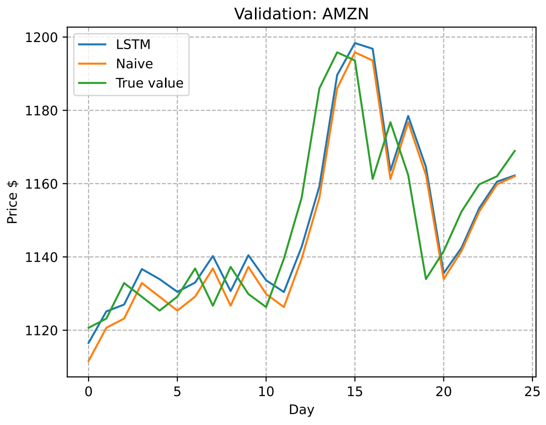
(b) Results for the AAPL stock with context



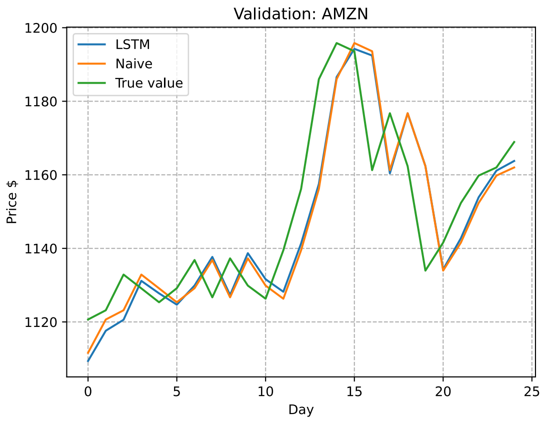
(c) Results for the AMD stock



(d) Results for the AMD stock with context



(e) Results for the AMZN stock



(f) Results for the AMZN stock with context

Figure 44: The first 25 time steps on the validation set

Figure 43 shows the first 25 time steps on the training data for an LSTM model using price and trendscore to predict the next price. The plots on the left are the results from a model **without** context while the results on the rights are from a model **with** context. Here, one effect of the context module is clear; In all of the plots related to the LSTM using context, the first predicted price is virtually completely accurate in relation to the true price. This means that the context module is trained to transform the stock identifier to a state that makes the LSTM capable of accurately predicting the first price. It also seems that the context module is superior when predicting the AMD stock until time step 14. When observing the AMZN stock, the context model seems to be superior until time step 10, and when observing the AAPL stock, the context model seems to only be superior until time step 2. It seems that the context module only contained information that could be utilized in the first time steps. This is augmented by the results in Figure 44 showing that this trend did not continue on to the validation data set. As the validation data set started at the time step 1328 there was no significant relationship between the context modules initial state and the predictions in the validation data set. There was no significant relationship found between the context module and the test set as well.

9.4 Using the same parameters across all stocks

Motivated by the investigation on whether stocks share similar characteristics that could be exploited to create a flexible and scalable stock prediction model, research question IV) was introduced. Additionally, this restricted the scope of this project into something manageable in the allocated time.

While the idea seemed interesting, and the LSTM configurations predicting all stocks actually achieved one result in terms of DA that was a statistically significant improvement, overall, this way of configuring the models did not produce results that were sufficient in the grand scheme of things. The main reasons for the performance to be unconvincing, at least in terms of actual usage, were the inconsistencies. The models predicting all stocks at the same time showed indication of struggling with keeping the results stable, since the individual results of each stock could vary significantly between runs, even when the configuration was entirely the same, as discussed in Section 9.1. Additionally, the configurations that achieved statistically significant improvements in terms of DA on the test set did not achieve the same on the validation set, showing another kind of inconsistency that is present. Overall, making the model more general seemed to give insufficient improvements when the cost of doing this, such as the increased complexity and time requirements, is considered. Considering that the general model did not manage to improve significantly on the regression metrics at all only adds reasons to this belief.

Experimentation with the ridge regression model suggested something similar, where the model was unable to use a universal α value on all stocks, producing results that were worse than the results of the naive model. Experimenting with different values for α did not give indication of improvements, only highlighting the challenges of attempting this.

The configurations examining each stock individually were not without flaws either. The ridge regression model with optimized α on the AAPL stock achieved improved performance over the naive model on the validation set, but failed to achieve the same on the test set, again showing inconsistent results across different time frames. The LSTM model experienced similar results, and while most of the results on the test set were improvements over the naive model, none were statistically significant over a random guessing model, which was shown in Section 9.2.

What seems to be the main reason causing the issues mentioned above is that the models are not able to extract information in the data that facilitates sufficiently accurate predictions that are consistent across stocks and time frames. Without data containing such information, configuring parameters, whether for a general model or a specialized model, does not seem viable as a means to improve performance.

9.5 Additional points

9.5.1 Different variance

There were found some evidence that indicated that there was a lack of generalization related to the general model. From Sections 9.1 to 9.4 it was found that the general model did not generalize well over different time frames. Many of the models that yielded good results on the validation set did poorly on the test set. This should be somewhat expected, as early stopping was used on the validation set. What was more unexpected was that the only model that was significantly better than the random models in relation to DA on the **test** set was worse than the naive model on the **validation** set.

In addition to the generalization issues, the problem of accurately predicting prices seemed to be very hard, as shown by the lack of convincing results. One aspect that can explain the difficulty of the problem is the varying variances across different stocks and different time frames, presented in Table 54.

Stock	Training set	Validation set	Test set
AAPL	0.0568	0.005449	0.05067
AMD	0.07422	0.01668	**0.1039**
AMZN	**0.07711**	0.04644	0.02954

BIDU	0.04855	0.005274	0.02766
DIS	0.07429	**0.002601**	**0.0017**
FB	0.06861	0.00389	0.0153
HD	0.06995	0.007464	0.0123
INTC	0.03981	0.02105	0.006823
KO	0.05121	0.0215	0.02188
NFLX	0.05901	**0.118**	0.04154
NVDA	0.05771	0.01111	0.07563
PFE	0.04892	0.002916	0.0219
QCOM	0.05225	0.01859	0.03321
TSLA	0.0592	0.004413	0.006116
WMT	**0.03121**	0.04524	0.01587
Variance of the variances	0.0001635	0.0008432	0.0007252

Table 54: Variance of the normalized prices. The bolded values are either the highest or lowest value in that column. The values in the bottom row are the variance of all values above. E.g. the value 0.0001635 related to the training set is $var(0.0568, 0.07422, ..., 0.03121)$

Table 54 shows that the variance in price is relatively stable in the training set across all stocks, the highest variance here being 0.07711 and the lowest being 0.03121. When looking at the validation set and the test set the variances vary much more, as seen in the bottom row. This can explain at least partly the difficulties associated with creating a general model that works for all stocks. As the models that predicted prices individually also did not yield results that were significantly better than the random guessing model, the difficulties seem to be more because of varying variance across different time frames and less because of the varying variances across the different stocks. As varying variances across different stocks very well could be an issue, although not as clearly as varying variances between time frames, it is recommended that further work should focus on single stocks or single stock indexes instead of multiple.

9.5.2 Overfitting on the data

One factor that should not be overlooked is the way LSTM is configured in this project does not make the models entirely resistant to overfitting. As were suggested from the results on the individual analysis of stocks, where the results on the validation set were noticeably better than the results on the test set. A reason for this could be that since the validation set is utilized to decide the early stopping point, a model could be more optimized at predicting the validation set compared to predicting another time frame, such as the test set. Essentially, the model is overfitted on the validation set. Another factor could be that since the validation set directly follows the training set in terms of time, these sets have more in common compared to the training set and the test set, making it easier to predict more accurately at the validation set than the test set. However, this was shown to not be the case in Section 9.5.1, at least not in terms of variance, as the variance in the test set was found to be more similar to the variance in the training set compared to the validation set. Another factor could be related to the lack of utilization of dropout, making overfitting more likely, especially when predicting one stock at a time due to the complexity being much lower compared to predicting all stocks at the same time.

9.5.3 Issues with lagging behind the actual values

Examining the plots together with the results, it became evident that the all models suffered from the same problem; not being able to deviate away from the current price, essentially meaning that the models did not perform far from what the 1-step-behind model was able to achieve. The issue with a model possessing tendencies with constantly lagging behind is that although the regression errors are relatively low, the direction accuracy of the predictions are rarely good enough for real life application. For instance, the naive model consistently achieves around 50% DA, too close to what have been observed to be the DA of random guessing. In short, such a model gives insufficient aids to investment strategies. The reason for this challenge with lagging behind seems to originate from the price direction between the current day and the day before having a low correlation with the price direction between the current day and the next day, suggesting the market to be extremely efficient. A model that is excessively dependent on using the current price as a basis for the predictions, and is not able to extract information from other data sources can therefore not achieve adequate performance.

9.5.4 Observations related to the evaluation method used

Table 55 shows the results of the naive model when predicting the next price. The top row has mainly been used in this thesis to evaluate the models, which is the averaged score of that particular column. Related to the other rows, one important observation is that there are often large disparities within the same metric for different stocks. This is especially apparent when observing the MSE results. Here, AMZN has the largest value of 1782 while

KO has the lowest value of 0.2959. The average MSE is much more affected by relative changes related to the AMZN stock than the KO stock. If the MSE related to the AMZN stock doubles, the average MSE will increase by $1782/15 = 118.8$ while if the MSE related to the KO stock doubles, the average MSE will only increase by $0.2959/15 = 0.01973$. The stocks that generally have higher prices will overshadow stocks that have low prices; there is clearly a positive correlation between MSE and the max price. If one only looks at the average MSE in order to evaluate the model, it is possible that multiple stocks that do very well are overlooked, simply because of their low stock price. This also applies to MAE, although to a lesser degree. The same correlation does not seem to exist between the max price and MAPE, but even though MAPE is not correlated to the max price, MAPE still varies, from AMD that has 3.377% to KO that has 0.7667%. This again means that KO might be left in the shadow when focusing on the average MAPE.

Stock	**MAPE**	**MAE**	**MSE**	**DA**	**Max price**
All	1.657%	4.428	143.2	51.47%	
AAPL	1.486%	2.798	15.46	56.36%	232.07
AMD	**3.377%**	0.7443	1.113	53.33%	32.72
AMZN	1.786%	**30.2**	**1782**	54.55%	2039.51
BIDU	1.747%	3.436	19.19	**43.03%**	271.45
DIS	0.8984%	1.005	1.963	**56.36%**	118.9
FB	1.705%	2.675	21.26	48.48%	217.5
HD	0.9772%	1.802	6.176	47.88%	213.85
INTC	1.479%	0.7026	0.9314	51.52%	53.94
KO	**0.7668%**	**0.3594**	**0.2959**	54.55%	50.51
NFLX	2.302%	7.457	94.67	47.27%	418.97
NVDA	2.498%	4.735	47.16	51.52%	289.36
PFE	1.014%	0.4307	0.3119	54.55%	46.23
QCOM	1.187%	0.7195	1.133	54.55%	75.09
TSLA	2.728%	8.501	154.9	46.67%	379.57
WMT	0.9004%	0.8587	1.587	51.52%	105.56

Table 55: The naive model predicting the next price

The DA measure does not seem to be affected to the same degree as the other metrics, mostly because the measure is not dependent on the magnitude of the prices. The takeaway from these observations is that when evaluating the model, one solution is to focus on the results related to the individual stocks rather than the averaged results. In relation to the metrics where the magnitude of the price plays a part, namely MAE and MSE, using the average results on the normalized values is another solution as the normalization forces the prices to be in the same magnitude.

9.5.5 Comparing against other works

As MAPE, MAE and MSE will vary significantly from stock to stock, a direct comparison with these metrics to other works that explores different stocks or indexes is infeasible. Therefore, the focus should be on direction accuracy.

Bollen, Mao & Zeng (2011) were able to predict the next Dow Jones Industrial Average close price by measuring general Twitter mood with a DA of 87.6%. This is clearly superior to the DA found in this thesis of 53.98%. This may be an indication that the general mood is a much better predictor of future stock prices than sentiment and trendscore related to specific stocks. The results may also indicate that the DJIA is more predictable than the stocks selected in this thesis, or more generally that stock market indexes are more predictable than individual stocks.

Jiahong Li, Bu & Wu (2017) achieved 50.71% direction accuracy when predicting the next day close price of the Chinese index CSI300 using sentiment extracted from forum posts. They achieved results similar to the results in this thesis even though they optimized direction accuracy. These results in addition to the results in this thesis provide evidence that predicting stock prices using sentiment from users on platforms directed to the stock market is a harder task than predicting prices using general mood. One possible explanation is that there is much more information from which mood can be extracted. Another hypothesis is that sentiment is decoupled from the traders own emotions and does not reflect their true feelings because of their ulterior motives related to making money on the stock market. People that own a stock have an incentive to motivate other people to buy the same stock, as this will increase the value of the stock. People may know this and are therefore not significantly affected by these sentiments. Further, Twitter is used for a vast number of topics, the financial market only being one of many others. The tweeters may not have the same ulterior motives when tweeting, and their tweets might to a higher degree reflect their true feelings. Therefore, the mood that Bollen, Mao & Zeng (2011) measured might be a more accurate representation of the users' emotions and is therefore more valuable when predicting the stock market.

Shynkevich, Coleman, Mcginnity & Belatreche (2015) reached a DA of up to 81.63% when predicting stocks using the news database LexisNexis. Their results were much better than the results presented in this thesis. It may be that it is the low credibility of tweets related to stocks and the high credibility of news articles that lead to the disparity between the results presented in this thesis and the results presented by Shynkevich, Coleman, Mcginnity & Belatreche (2015).

10 Conclusion

This project derived from the idea that combining different data sources could improve predictions of stock market prices, leading to a goal of investigating whether easily accessible data could contribute to more accurate stock price predictions using machine learning. Four research questions arose from this overarching goal, and the experiments conducted aimed at answering these questions.

To answer all the research questions, especially research question I), data from several sources were collected, analyzed and preprocessed as deemed necessary. Historical stock data were gathered from Investing.com, to represent the trading and financial aspect. Data provided by StockFluence were utilized to represent the sentiment aspect. Lastly, Google Trends was used to get access to search popularity data, to represent another feature of the market.

Generally, the results suggested that, in terms of practical usage and achieving considerably better performance compared to random guessing, the data available contributed insufficiently to boosting performance on stock market price predictions. All configurations had issues, either with unconvincing results, inconsistent results, or that the improvements were too small to be considered for real life application. If improvements could be made on one time frame, there seemed to be no guarantee that similar results could be achieved on another time frame. Also, a configuration showing improvements on one stock did not necessarily mean that the same configuration would attain improved results on another stock. In short, the data gathered in this project did not manage to facilitate accurate and usable price predictions, at least not with the models experimented with. The main reason seemed to be due to the differences, mainly between time frames, but also between stocks. A representative measure for this issue is the variance, which showed that the variance between stocks, and across time frames, could vary to a large degree.

Since the data did not contribute with usable data in terms of prediction, performance differences between the models were small in terms of the quantitative evaluation metrics. Although the LSTM model achieved one result that was significantly better than random guessing in terms of direction accuracy, a feat that was not obtainable by the other models, the inconsistent performance on different time frames makes it hard to recommend such a model for practical application, as of now. The results suggested that there are LSTM configurations that are able to outperform the baseline models in terms of the evaluation metrics, but such configurations are not able to accomplish this consistently.

With an unfavorable starting point with the LSTM models exhibiting these particular issues, including a context module did not seem to contribute in any meaningful way, with improvements being marginal at best. What the context module did for the most part was to improve the initial predictions on the training set, with little actual improvements on the validation set and the test set. This again seems to happen as a result of the data not providing information that can assist the model in deviating far enough away from the current price when predicting the next price, or deviating enough from 0 (or the mean) when predicting the price change. Overall, including the context module did little to improve the issues the LSTM model was facing.

Lastly, discussing whether a more general model using the same parameters across all stocks was challenging. The available data seemed to provide little positive impact on prediction, as if the data did not contain information that could make the predictions usable. This made it hard to confidently recommend one configuration over the other, as all models and configurations exhibited similar tendencies with inconsistent performance. With the data gathered in this project, it is, as of now, not possible to conclude that a model with the same parameters across all stocks is able to outperform the baseline models. It is also not possible to conclude whether a general model is better than a specialized model (that has all parameters identical for every stock), as similar issues seem to affect both configurations. What can be said is that the less general a model is, and the less complex, the easier it has been to troubleshoot what the origin of the issues faced is.

The answer to the overarching goal of this project is that it is not possible to better the accuracy of stock prices prediction with easily accessible data, at least not with the data available in this project, and not with the machine learning models experimented with. Improvements, if any, are not consistent across stocks and time frames, making real life application challenging.

11 Future work

In this chapter, insights gathered throughout this project regarding further approaches on this field are presented. These are based on the results presented in Chapter 8 and discussion from Chapter 9, in addition to the challenges that were faced.

Predicting the volatility instead of the price

As the volatility of a stock reflects the risks related to the stock, being able to accurately predict it is advantageous when financial gain is the goal. Investopedia.com, 2020 lists multiple strategies to profit from high volatility.

Optimising direction accuracy

In order to improve the direction accuracy, optimizing the direction accuracy instead of the regression metrics as done in this thesis seems to be a viable option.

Combine prediction with a realistic trading strategy

To better investigate the practical usability of the models implemented, including a realistic trading strategy that can be used on the stock market or a simulation of a stock market should be done. In this way, assessing if the models are able to achieve financial gain or not, which ultimately is the goal, is possible. Also, it is possible to experiment with multiple trading strategies to determine which one is more suitable for each model. Investigating the practical usage of a model might also reveal some interesting behavior not otherwise visible from performance measures or visualization alone.

Implement reinforcement learning

It could be interesting to see how reinforcement learning coupled with deep learning can achieve in terms of financial success. Putting emphasis on your previous actions and learn from them could make tasks easier to overcome. Predicting stock prices and achieving monetary gain could be one of these task. Combine with the already mentioned points of improvements, this could prove to be a viable option.

Investigate other data sources

Although the included data sources did not manage to significantly improve the performance on this task, other data sources or combinations of other data sources could prove to be more successful. As lots of data are already easily available, and more seem to become available, this might be worth the investigation and resources required. Combined with the previously mentioned suggestions for further work, some interesting information or behavior can be revealed. Comparing performance improvements from publicly or easily available data to data that could be considered more exclusive, for instance data that are only accessible with premium memberships or within closed groups, could most definitely be an interesting investigation resulting in fascinating discoveries.

References

[1] Jethin Abraham, Daniel Higdon, John Nelson & Juan Ibarra. «Cryptocurrency Price Prediction Using Tweet Volumes and Sentiment Analysis» (2018).

[2] Abdulaziz M. Alayba, Vasile Palade, Matthew England & Rahat Iqbal. «A Combined CNN and LSTM Model for Arabic Sentiment Analysis» (2018).

[3] Sivan Alon, Simon Perrigaud & Meredith Neyrand. «Predicting American Idol with Twitter Sentiment» (2013).

[4] Khaled A. Althelaya, El-Sayed M. El-Alfy & Salahadin Mohammed. «Evaluation of Bidirectional LSTM for Short- and Long-Term Stock Market Prediction» (2018).

[5] Julio Amador, Sofia Collignon-Delmar, Kenneth Benoit & Akitaka Matsuo. «Predicting the Brexit Vote by Tracking and Classifying Public Opinion Using Twitter Data» (2017).

[6] Franz Bewerunge. *Google Trends: How to acquire daily data for broad time frames*. 2018. URL: https://medium.com/@bewerunge.franz/google-trends-how-to-acquire-daily-data-for-broad-time-frames-b6c6dfe200e6 (visited on 04/20/2020).

[7] Johan Bollen, Huina Mao & Xiaojun Zeng. «Twitter mood predicts the stock market». *Journal of Computational Science* 2.1 (2011), pp. 1–8. ISSN: 1877-7503. DOI: https://doi.org/10.1016/j.jocs.2010.12.007. URL: http://www.sciencedirect.com/science/article/pii/S187775031100007X.

[8] Salah Bouktif, Ali Fiaz, Ali Ouni & Adel Serhani. «Optimal Deep Learning LSTM Model for Electric Load Forecasting using Feature Selection and Genetic Algorithm: Comparison with Machine Learning Approaches» (2018).

[9] Jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (visited on 07/02/2020).

[10] Guillaume Chevalier. *The LSTM cell.png*. URL: https://commons.wikimedia.org/wiki/File:The_LSTM_cell.png#Licensing (visited on 07/07/2020).

[11] C. Cortes & V. Vapnik. *Machine learning 20* (1995), pp. 273–297. URL: https://link.springer.com/article/10.1007/BF00994018.

[12] Imane El Alaoui, Youssef Gahi, Rochdi Messoussi, Youness Chaabi, Alexis Todoskoff & Abdessamad Kobi. «A novel adaptable approach for sentiment analysis on big social data» (2018).

[13] Eugene F. Fama. «Efficient Capital Markets: A Review of Theory and Empirical Work». *The Journal of Finance* 25.2 (1970), pp. 383–417. ISSN: 00221082, 15406261. URL: http://www.jstor.org/stable/2325486.

[14] Thomas Fischer & Christopher Krauss. «Deep learning with long short-term memory networks for financial market predictions». *European Journal of Operational Research* 270.2 (2018), pp. 654–669. ISSN: 0377-2217. DOI: https://doi.org/10.

1016/j.ejor.2017.11.054. URL: http://www.sciencedirect.com/science/article/pii/S0377221717310652.

[15] Svitlana Galeshchuk, Oleksandra Vasylchyshyn & Andriy Krysovatyy. «Bitcoin Response to Twitter Sentiments» (2018).

[16] Andre Gensler, Janosch Henze, Bernhard Sick & Nils Raabe. «Deep Learning for Solar Power Forecasting – An Approach Using Autoencoder and LSTM Neural Networks» (2016).

[17] Ian Goodfellow, Yoshua Bengio & Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[18] C. W. J. Granger. «Investigating Causal Relations by Econometric Models and Cross-spectral Methods». *Econometrica* 37.3 (1969), pp. 424–438. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1912791.

[19] Jiawei Han, Micheline Kamber & Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.

[20] Internetlivestats.com. *Internetlivestats.com*. URL: https://www.internetlivestats.com/google-search-statistics/ (visited on 06/17/2019).

[21] Investing.com. *Investing.com - Stock Market Quotes & Financial News*. URL: https://www.investing.com/ (visited on 02/13/2020).

[22] Investopedia.com. *Investopedia.com*. URL: https://www.investopedia.com/slide-show/worlds-greatest-investors/ (visited on 06/21/2020).

[23] Bhumika M. Jadav & Vimalkumar B. Vaghela. «Sentiment Analysis using Support Vector Machine based on Feature Selection and Semantic Analysis» (2016).

[24] Arti Jain, Shashank Tripathi, Harsh DwarDwivedi & Pranav Saxena. «Forecasting Price of Cryptocurrencies Using Tweets Sentiment Analysis» (2018).

[25] Keras. *Keras*. URL: https://keras.io/ (visited on 07/24/2020).

[26] Diederik P. Kingma & Jimmy Lei Ba. «Adam: A Method for Stochastic Optimization» (2015). URL: https://arxiv.org/pdf/1412.6980.pdf.

[27] John Kordonis, Symeon Symeonidis & Avi Arampatzis. «Stock Price Forecasting via Sentiment Analysis on Twitter» (2016).

[28] Alex Krizhevsky, Ilya Sutskever & Geoffrey E Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou & K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[29] Gunjan Kumar. «Machine Learning for Soccer Analytics» (2013).

[30] Jiahong Li, Hui Bu & Junjie Wu. «Sentiment-Aware Stock Market Prediction: A Deep Learning Method» (2017).

[31] Jian Li, Zhenjing Xu, Lean Yu & Ling Tang. «Forecasting oil price trends with sentiment of online news articles» (2016).

[32] Burton G. Malkiel. «The Efficient Market Hypothesis and Its Critics». *Journal of Economic Perspectives* 17.1 (Mar. 2003), pp. 59–82. DOI: 10.1257/089533003321164958. URL: http://www.aeaweb.org/articles?id=10.1257/089533003321164958.

[33] Amir Mosavi, Pinar Ozturk & Kwok-wing Chau. «Flood Prediction Using Machine Learning Models: Literature Review» (2018).

[34] Sima Siami Namin & Akbar Siami Namin. «Forecasting economic and financial time series: ARIMA vs. LSTM» (2018).

[35] Patrick Andre Næss. «Investigation of Multivariate Freight Rate Prediction Using Machine Learning and AIS Data» (2018).

[36] Peter Norvig & Stuart J. Russel. *Artificial Intelligence: A Modern Approach*. Pearson, 2016.

[37] NumPy. *NumPy*. URL: https://www.numpy.org/ (visited on 07/01/2020).

[38] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 11/11/2018).

[39] Christopher Olah. *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 07/07/2020).

[40] Preety & Sunny Dahiya. «Sentinement Analysis Using SVM and Naive Bayes Algorithm» (2015).

[41] Tobias Preis, Helen Susannah Moat & H. Eugene Stanley. «Quantifying Trading Behavior in Financial Markets Using Google Trends». *Scientific Reports* 3 (Apr. 2013). URL: https://doi.org/10.1038/srep01684.

[42] Gabriele Ranco, Darko Aleksovski, Guido Caldarelli, Miha Grčar & Igor Mozetič. «The Effects of Twitter Sentiment on Stock Price Returns». *PloS one* 10 (Sept. 2015), e0138441. DOI: 10.1371/journal.pone.0138441.

[43] David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams. «Learning representations by back-propagating errors» (1986).

[44] Pritee Salunkhe & Sachin Deshmukh. «Twitter Based Election Prediction and Analysis» (2017).

[45] Scientificamerican.com. *Scientificamerican.com*. URL: https://www.scientificamerican.com/podcast/episode/computers-go-head-to-head-with-humans-on-face-recognition/ (visited on 05/15/2020).

[46] Scikit-Learn. *Scikit-learn*. URL: https://scikit-learn.org/ (visited on 06/30/2020).

[47] Parul Sharma & Teng-Sheng Moh. «Prediction of Indian election using sentiment analysis on Hindi Twitter» (2016).

[48] Yauheniya Shynkevich, Sonya Coleman, T.M. Mcginnity & Ammar Belatreche. «Stock Price Prediction based on Stock-Specific and Sub-Industry-Specific News Articles». In: July 2015. DOI: 10.1109/IJCNN.2015.7280517.

[49] Pedro M. Sosa. «Twitter Sentiment Analysis using combined LSTM-CNN Models» (2017).

[50] Statista.com. *Statista.com*. URL: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/ (visited on 05/15/2020).
[51] statsmodels.org. *statsmodels.org*. URL: https://www.statsmodels.org/stable/index.html (visited on 07/07/2020).
[52] Timo Stöttner. *Why Data should be Normalized before Training a Neural Network*. URL: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d (visited on 07/25/2020).
[53] StockFluence. *StockFluence | Stock Sentiment Analysis*. URL: https://www.stockfluence.com/ (visited on 02/13/2020).
[54] TensorFlow. *TensorFlow*. URL: https://www.tensorflow.org/ (visited on 07/24/2020).
[55] Google Trends. *Google Trends*. URL: https://trends.google.com (visited on 03/05/2020).
[56] Ian H. Witten, Eibe Frank & Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.
[57] Ruoxuan Xiong, Eric P. Nichols & Yuan Shen. «Deep Learning Stock Volatility with Google Domestic Trends» (2016).
[58] Xin Yan & Xiao Gang Su. *Linear Regression Analysis: Theory and Computing*. 5 Toh Tuck Link, Singapore: World Scientific, 2009.
[59] Kristin Yeager. *SPSS TUTORIALS: PEARSON CORRELATION*. June 2020. URL: https://libguides.library.kent.edu/SPSS/PearsonCorr.
[60] Lei Zhang, Shuai Wang & Bing Liu. «Deep Learning for Sentiment Analysis: A Survey» (2018).

Appendix

A Details on all stocks

Stock	Name	Description
AAPL	Apple	A technology company focusing on consumer electronics and software. Founded in 1976.
AMD	Advanced Micro Devices	A technology company developing computer processors and related technologies for both consumers and businesses. Founded in 1969.
AMZN	Amazon.com	A technology company focusing on e-commerce, cloud computing and artificial intelligence. Founded in 1994.
BIDU	Baidu	A techonology company focusing on artificial intelligence and internet-related services and products. Founded in 2000.
DIS	The Walt Disney Company	A conglomerate focusing on mass media and entertainment. Founded in 1923.
FB	Facebook	A technology company focusing on social media and social networking services. Founded in 2004.
HD	The Home Depot	A retailer company focusing on home improvement. Founded in 1978.
INTC	Intel	A technology company focusing on manufacturing semiconductor chips and related technologies. Founded in 1968.
KO	The Coca-Cola Company	A company focusing on non-alcoholic beverages. Founded in 1886.
NFLX	Netflix	A company focusing on providing media services. Founded in 1997.

NVDA	Nvidia	A technology company focusing on making of graphics processing units and system on a chip units. Founded in 1993.
PFE	Pfizer	A pharmaceutical company. Founded in 1849.
QCOM	Qualcomm	A technology company focusing on semiconductor and telecommunications equipment. Founded in 1985.
TSLA	Tesla	An automotive and energy company focusing on electric cars. Founded in 2003.
WMT	Walmart	A retail corporation focusing on hypermarkets, discount department stores and grocery stores. Founded in 1962.

Table 56: The stocks included in the project.

B Results

B.1 Hyperparameter search

{Dropout, Layer size, Loss }	**MAPE**	**MAE**	**MSE**	**DA**
0.0, [160], mae				
0	1.388%	3.285	56.6	50.75%
1	1.399%	3.271	55.6	51.15%
2	1.399%	3.386	60.37	50.38%
Mean	1.396%	3.314	57.52	50.76%
Mean Rank	1	1	1	4
Sum rank	7			
0.0, [128], mae				
0	1.405%	3.331	58.43	50.34%
1	1.418%	3.365	58.82	49.01%
2	1.386%	3.291	57.75	51.52%

Mean	1.403%	3.329	58.34	50.29%
Mean Rank	2	2	2	6
Sum rank	12			
0.2, [128], mae				
0	1.412%	3.576	69.58	50.22%
1	1.417%	3.67	74.62	49.33%
2	1.389%	3.413	61.66	50.63%
Mean	1.406%	3.553	68.62	50.06%
Mean Rank	3	3	3	10
Sum rank	19			
0.0, [32], mae				
0	1.463%	3.966	89.39	50.95%
1	1.444%	3.392	57.2	51.6%
2	1.426%	3.42	59.5	49.33%
Mean	1.444%	3.593	68.7	50.63%
Mean Rank	7	4	4	5
Sum rank	20			
0.2, [160], mae				
0	1.419%	3.714	76.75	50.79%
1	1.388%	3.439	63.64	49.33%
2	1.427%	3.752	78.54	50.63%
Mean	1.411%	3.635	72.98	50.25%
Mean Rank	4	6	7	8
Sum rank	25			
0.5, [160], mae				
0	1.424%	3.625	71.11	49.41%
1	1.407%	3.51	65.66	50.06%
2	1.419%	3.665	72.79	49.98%
Mean	1.417%	3.6	69.85	49.82%
Mean Rank	5	5	5	13
Sum rank	28			
0.0, [128], mse				

0	1.447%	3.523	63.73	50.67%
1	1.46%	3.665	70.27	50.71%
2	1.453%	3.825	78.72	49.21%
Mean	1.454%	3.671	70.91	50.2%
Mean Rank	8	7	6	9
Sum rank	30			
0.5, [128], mae				
0	1.468%	4.034	94.61	49.74%
1	1.425%	3.716	75.23	49.74%
2	1.394%	3.417	61.78	50.42%
Mean	1.429%	3.722	77.21	49.97%
Mean Rank	6	8	8	11
Sum rank	33			
0.0, [32], mse				
0	1.475%	3.963	89.55	50.95%
1	1.468%	3.817	78.82	50.95%
2	1.493%	4.035	90.96	50.95%
Mean	1.478%	3.938	86.44	50.95%
Mean Rank	11	11	10	3
Sum rank	35			
0.2, [128], mse				
0	1.487%	4.184	107.0	50.59%
1	1.544%	4.434	123.1	51.03%
2	1.4%	3.328	59.19	51.92%
Mean	1.477%	3.982	96.42	51.18%
Mean Rank	10	12	12	1
Sum rank	35			
0.0, [160], mse				
0	1.495%	4.177	100.8	49.29%
1	1.477%	3.551	63.25	51.23%
2	1.472%	3.758	74.93	50.26%
Mean	1.481%	3.828	79.65	50.26%

Mean Rank	12	9	9	7
Sum rank	37			
0.5, [128], mse				
0	1.501%	4.321	115.2	49.54%
1	1.483%	4.086	97.0	50.02%
2	1.389%	3.26	54.77	50.22%
Mean	1.458%	3.889	88.98	49.93%
Mean Rank	9	10	11	12
Sum rank	42			
0.2, [160], mse				
0	1.54%	4.61	141.4	51.07%
1	1.477%	4.06	99.81	51.11%
2	1.528%	4.259	108.2	51.31%
Mean	1.515%	4.31	116.5	51.16%
Mean Rank	15	15	15	2
Sum rank	47			
0.5, [160], mse				
0	1.508%	4.363	117.4	49.58%
1	1.463%	3.964	89.68	49.58%
2	1.485%	4.166	103.0	50.63%
Mean	1.485%	4.164	103.3	49.93%
Mean Rank	13	13	13	12
Sum rank	51			
0.2, [32], mse				
0	1.617%	5.103	165.6	48.24%
1	1.427%	3.518	65.09	49.01%
2	1.499%	4.277	108.5	48.81%
Mean	1.514%	4.299	113.1	48.69%
Mean Rank	14	14	14	14
Sum rank	56			
0.2, [32], mae				
0	1.594%	5.014	164.2	48.48%

1	1.547%	4.614	130.4	48.57%
2	1.492%	4.253	106.6	48.04%
Mean	1.544%	4.627	133.7	48.36%
Mean Rank	16	16	16	15
Sum rank	63			
0.5, [32], mae				
0	1.805%	6.045	236.7	47.76%
1	1.867%	6.456	280.1	47.84%
2	1.63%	5.048	155.3	48.81%
Mean	1.768%	5.849	224.0	48.13%
Mean Rank	17	17	17	16
Sum rank	67			
0.5, [32], mse				
0	1.935%	6.862	329.8	48.48%
1	1.804%	5.852	207.2	47.84%
2	1.808%	6.091	242.9	47.39%
Mean	1.849%	6.268	260.0	47.91%
Mean Rank	18	18	18	17
Sum rank	71			

Table 57: Full table: Hyperparameter search

B.2 Issues with the bidirectional implementation

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, open, high, low, volume, direction, change, BidirLSTM, [54, 53, 53]				
0	4.965%	7.984	582.5	49.44%
1	6.587%	9.831	642.0	49.57%
2	8.635%	11.35	680.3	48.72%
Mean	6.729%	9.721	634.9	49.25%
Mean Rank	16	16	16	16

Sum rank	64			
price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, BidirLSTM, [54, 53, 53]				
0	5.985%	7.171	174.9	48.88%
1	11.26%	16.83	1331	47.39%
2	6.925%	11.7	1043	49.03%
Mean	8.055%	11.9	850.2	48.43%
Mean Rank	17	17	17	18
Sum rank	69			
price, trendscore, BidirLSTM, [54, 53, 53]				
0	4.667%	6.806	326.6	48.99%
1	15.79%	25.02	2707	47.09%
2	7.384%	12.1	1021	48.48%
Mean	9.28%	14.64	1351	48.19%
Mean Rank	18	18	18	19
Sum rank	73			
price, BidirLSTM, [54, 53, 53]				
0	4.108%	5.532	190.7	49.92%
1	4.993%	8.26	528.2	49.2%
2	21.46%	34.82	5059	47.16%
Mean	10.19%	16.2	1926	48.76%
Mean Rank	19	19	19	17
Sum rank	74			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, BidirLSTM, [54, 53, 53]				
0	18.49%	29.01	3359	47.05%
1	11.2%	17.0	1485	47.37%
2	9.211%	13.16	1056	49.29%

Mean	12.97%	19.72	1966	47.9%
Mean Rank	20	20	20	20
Sum rank	80			
price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, BidirLSTM, [80, 80]				
0	10.84%	16.0	1319	48.74%
1	18.37%	27.71	2818	47.36%
2	18.68%	28.88	3047	47.22%
Mean	15.97%	24.2	2395	47.77%
Mean Rank	21	21	21	21
Sum rank	84			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, BidirLSTM, [80, 80]				
0	13.51%	18.93	1190	47.19%
1	24.4%	40.71	7401	47.05%
2	13.01%	19.54	1591	47.15%
Mean	16.97%	26.39	3394	47.13%
Mean Rank	22	22	22	22
Sum rank	88			
price, trendscore, BidirLSTM, [80, 80]				
0	27.98%	45.15	8362	47.03%
1	19.44%	30.28	3617	47.12%
2	16.97%	25.89	2833	47.2%
Mean	21.47%	33.77	4937	47.12%
Mean Rank	24	24	24	23
Sum rank	95			
price, open, high, low, volume, direction, change, BidirLSTM, [80, 80]				

0	19.14%	29.03	3071	47.12%
1	22.41%	36.03	5407	47.03%
2	17.48%	25.89	2446	46.98%
Mean	19.68%	30.32	3641	47.04%
Mean Rank	23	23	23	28
Sum rank	97			
price, BidirLSTM, [80, 80]				
0	16.99%	25.62	2319	47.1%
1	33.99%	58.12	15707	47.05%
2	27.4%	45.98	9393	47.06%
Mean	26.12%	43.24	9140	47.07%
Mean Rank	25	25	25	26
Sum rank	101			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, BidirLSTM, [160]				
0	24.47%	42.1	8860	47.16%
1	25.38%	43.49	9187	47.03%
2	29.2%	50.06	11842	47.05%
Mean	26.35%	45.22	9963	47.08%
Mean Rank	26	26	26	25
Sum rank	103			
price, open, high, low, volume, direction, change, BidirLSTM, [160]				
0	30.37%	52.03	12605	47.08%
1	28.06%	48.48	11331	47.12%
2	28.51%	49.02	11333	47.08%
Mean	28.98%	49.84	11756	47.09%
Mean Rank	27	27	27	24
Sum rank	105			

price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, BidirLSTM, [160]				
0	43.34%	74.6	26780	47.1%
1	40.3%	69.23	22951	47.06%
2	44.59%	76.62	28001	47.08%
Mean	42.74%	73.48	25911	47.08%
Mean Rank	28	28	28	25
Sum rank	109			
price, trendscore, BidirLSTM, [160]				
0	41.96%	72.22	24804	47.06%
1	44.62%	76.64	27742	47.09%
2	44.66%	76.76	27768	47.05%
Mean	43.75%	75.21	26771	47.07%
Mean Rank	29	29	29	27
Sum rank	114			
price, BidirLSTM, [160]				
0	44.27%	76.15	27360	47.03%
1	43.28%	74.31	25923	47.03%
2	44.59%	76.7	27799	47.05%
Mean	44.05%	75.72	27027	47.04%
Mean Rank	30	30	30	29
Sum rank	119			

Table 58: Bidirectional

B.3 Feature analysis, predicting price

Seed	MAPE	MAE	MSE	DA
price, [160]				
0	1.352%	3.168	52.68	52.4%
1	1.351%	3.134	51.83	52.61%

2	1.353%	3.17	53.06	52.4%
Mean	1.352%	3.157	52.52	52.47%
Mean Rank	1	2	2	2
Sum rank	7			
price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.362%	3.163	52.58	52.57%
1	1.347%	3.141	52.78	52.69%
2	1.351%	3.16	52.55	52.48%
Mean	1.353%	3.155	52.63	52.58%
Mean Rank	2	1	3	1
Sum rank	7			
price, trendscore, [160]				
0	1.351%	3.137	52.15	52.57%
1	1.36%	3.2	52.94	50.95%
2	1.349%	3.137	51.43	51.84%
Mean	1.354%	3.158	52.18	51.78%
Mean Rank	3	3	1	5
Sum rank	12			
price, trendscore, [80, 80]				
0	1.412%	3.35	57.26	51.23%
1	1.365%	3.18	52.73	52.16%
2	1.369%	3.242	54.61	52.73%
Mean	1.382%	3.257	54.87	52.04%
Mean Rank	6	4	4	3
Sum rank	17			
price, [80, 80]				
0	1.387%	3.378	59.26	52.12%
1	1.363%	3.237	54.71	51.68%
2	1.375%	3.22	53.81	52.04%
Mean	1.375%	3.278	55.93	51.95%

Mean Rank	4	5	5	4
Sum rank	18			
price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [80, 80]				
0	1.389%	3.34	58.4	51.84%
1	1.359%	3.219	55.02	51.52%
2	1.383%	3.312	56.75	51.8%
Mean	1.377%	3.29	56.73	51.72%
Mean Rank	5	6	6	7
Sum rank	24			
price, open, high, low, volume, direction, change, [160]				
0	1.428%	3.411	61.22	50.75%
1	1.369%	3.252	57.55	50.55%
2	1.383%	3.259	57.51	51.84%
Mean	1.393%	3.307	58.76	51.04%
Mean Rank	7	7	8	11
Sum rank	33			
price, [54, 53, 53]				
0	1.39%	3.333	57.26	52.24%
1	1.424%	3.495	63.79	51.43%
2	1.431%	3.667	71.47	51.52%
Mean	1.415%	3.498	64.17	51.73%
Mean Rank	10	10	10	6
Sum rank	36			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	1.388%	3.285	56.6	50.75%
1	1.399%	3.271	55.6	51.15%

2	1.399%	3.386	60.37	50.38%
Mean	1.396%	3.314	57.52	50.76%
Mean Rank	8	8	7	13
Sum rank	36			
price, open, high, low, volume, direction, change, [80, 80]				
0	1.411%	3.597	71.65	49.66%
1	1.381%	3.326	58.26	51.76%
2	1.395%	3.278	59.69	51.23%
Mean	1.396%	3.4	63.2	50.88%
Mean Rank	9	9	9	12
Sum rank	39			
price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [54, 53, 53]				
0	1.518%	4.335	112.9	51.35%
1	1.461%	3.538	64.22	51.52%
2	1.503%	4.253	108.2	51.11%
Mean	1.494%	4.042	95.14	51.33%
Mean Rank	13	12	12	9
Sum rank	46			
price, trendscore, [54, 53, 53]				
0	1.519%	4.378	118.0	50.83%
1	1.473%	4.063	106.2	51.88%
2	1.458%	3.899	83.89	51.47%
Mean	1.483%	4.113	102.7	51.39%
Mean Rank	12	13	13	8
Sum rank	46			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [80, 80]				

0	1.452%	3.856	84.34	51.03%
1	1.452%	3.807	80.1	50.18%
2	1.479%	3.834	77.9	49.66%
Mean	1.461%	3.833	80.78	50.29%
Mean Rank	11	11	11	15
Sum rank	48			
price, open, high, low, volume, direction, change, [54, 53, 53]				
0	1.716%	5.638	235.6	50.59%
1	1.517%	4.126	98.26	51.88%
2	1.53%	4.343	114.7	51.39%
Mean	1.588%	4.703	149.5	51.29%
Mean Rank	14	14	14	10
Sum rank	52			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [54, 53, 53]				
0	1.665%	5.267	194.6	49.86%
1	1.683%	5.597	229.4	50.87%
2	1.708%	5.502	218.9	50.38%
Mean	1.685%	5.456	214.3	50.37%
Mean Rank	15	15	15	14
Sum rank	59			

Table 59: Full table: Feature search using vanilla LSTMs and stacked LSTMs

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, [160]				
0	1.352%	3.141	52.43	52.81%
1	1.348%	3.146	53.03	52.48%
2	1.35%	3.141	53.19	52.85%

3	1.351%	3.151	52.78	52.44%
4	1.348%	3.153	53.17	51.52%
5	1.349%	3.141	52.33	52.32%
6	1.346%	3.129	52.66	52.57%
7	1.355%	3.169	52.41	52.4%
8	1.346%	3.133	52.72	52.28%
9	1.347%	3.136	52.39	52.36%
Mean	1.349%	3.144	52.71	52.4%
Mean Rank	1	1	2	1
Sum rank	5			
price, trendscore, [160]				
0	1.355%	3.175	52.73	52.04%
1	1.355%	3.139	51.65	52.0%
2	1.361%	3.188	53.56	52.36%
3	1.351%	3.156	52.46	52.4%
4	1.368%	3.243	54.97	51.68%
5	1.347%	3.119	51.15	53.9%
6	1.345%	3.118	51.9	52.53%
7	1.355%	3.162	52.63	52.32%
8	1.354%	3.122	51.11	52.12%
9	1.348%	3.133	51.79	51.76%
Mean	1.354%	3.155	52.4	52.31%
Mean Rank	2	2	1	2
Sum rank	7			
price, open, high, low, volume, direction, change, [160]				
0	1.374%	3.245	57.63	50.67%
1	1.362%	3.225	57.55	50.91%
2	1.375%	3.289	58.7	49.25%
3	1.371%	3.236	57.68	49.29%
4	1.377%	3.238	57.48	50.55%
5	1.392%	3.305	56.67	51.92%
6	1.38%	3.275	58.19	50.79%
7	1.378%	3.24	58.78	51.31%

8	1.368%	3.228	56.16	51.07%
9	1.38%	3.335	60.32	50.06%
Mean	1.376%	3.262	57.92	50.58%
Mean Rank	3	3	3	3
Sum rank	12			

Table 60: Full table: Trendscore in addition to price

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, positive_prop, negative_prop, neutral_prop, [160]				
0	1.349%	3.139	53.06	52.77%
1	1.349%	3.156	53.41	52.73%
2	1.349%	3.184	53.89	52.36%
3	1.351%	3.162	53.96	51.8%
4	1.348%	3.144	52.94	52.36%
5	1.35%	3.19	54.58	51.19%
6	1.346%	3.141	53.33	53.01%
7	1.35%	3.188	54.05	52.4%
8	1.344%	3.144	53.22	52.65%
9	1.343%	3.141	52.35	52.44%
Mean	1.348%	3.159	53.48	52.37%
Mean Rank	1	1	2	1
Sum rank	5			
price, positive, negative, neutral, [160]				
0	1.363%	3.214	53.73	51.52%
1	1.362%	3.233	54.74	52.04%
2	1.357%	3.234	54.9	51.88%
3	1.356%	3.138	51.72	52.32%
4	1.361%	3.184	52.55	52.0%
5	1.364%	3.166	51.42	53.17%
6	1.355%	3.149	52.24	50.51%
7	1.347%	3.133	52.21	51.8%

8	1.36%	3.206	53.67	52.36%
9	1.346%	3.139	52.15	52.0%
Mean	1.357%	3.18	52.93	51.96%
Mean Rank	2	2	1	2
Sum rank	7			
price, volume, direction, change, [160]				
0	1.358%	3.17	52.63	51.64%
1	1.356%	3.149	51.9	50.95%
2	1.398%	3.354	59.42	52.28%
3	1.344%	3.137	53.32	53.05%
4	1.369%	3.393	64.57	51.52%
5	1.358%	3.241	55.83	52.08%
6	1.388%	3.326	57.54	52.24%
7	1.369%	3.371	61.6	52.77%
8	1.356%	3.145	52.87	51.6%
9	1.375%	3.182	54.48	50.99%
Mean	1.367%	3.247	56.42	51.91%
Mean Rank	4	3	3	3
Sum rank	13			
price, open, high, low, [160]				
0	1.37%	3.291	57.15	51.23%
1	1.384%	3.37	60.12	51.03%
2	1.362%	3.246	55.31	50.26%
3	1.361%	3.274	57.24	50.42%
4	1.364%	3.237	55.78	50.22%
5	1.351%	3.191	54.75	51.19%
6	1.363%	3.261	56.93	50.46%
7	1.364%	3.242	55.96	50.71%
8	1.368%	3.251	56.2	50.75%
9	1.365%	3.244	55.49	50.59%
Mean	1.365%	3.261	56.5	50.69%
Mean Rank	3	4	4	4
Sum rank	15			

Table 61: Full table: Dividing into smaller feature sets

B.4 Introducing the context module

Seed	MAPE	MAE	MSE	DA
price, trendscore, [160]				
0	1.347%	3.135	51.66	52.85%
1	1.348%	3.143	51.89	52.48%
2	1.345%	3.109	51.25	52.24%
Mean	1.347%	3.129	51.6	52.53%
Mean Rank	1	1	1	2
Sum rank	5			
price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.349%	3.157	53.32	52.08%
1	1.348%	3.143	52.91	53.05%
2	1.356%	3.148	52.94	52.81%
Mean	1.351%	3.149	53.06	52.65%
Mean Rank	3	2	3	1
Sum rank	9			
price, [160]				
0	1.347%	3.15	52.75	52.24%
1	1.348%	3.144	52.21	52.93%
2	1.352%	3.192	53.47	50.71%
Mean	1.349%	3.162	52.81	51.96%
Mean Rank	2	3	2	3
Sum rank	10			

price, open, high, low, volume, direction, change, [160]				
0	1.375%	3.224	56.94	50.26%
1	1.375%	3.247	57.45	49.86%
2	1.371%	3.238	57.15	49.82%
Mean	1.373%	3.237	57.18	49.98%
Mean Rank	4	4	4	5
Sum rank	17			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	1.433%	3.367	59.51	51.52%
1	1.405%	3.396	59.99	49.62%
2	1.422%	3.585	68.79	50.14%
Mean	1.42%	3.449	62.76	50.42%
Mean Rank	5	5	5	4
Sum rank	19			

Table 62: Full table: Predicting the next price using the context module.

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, trendscore, [160]				
0	1.343%	3.126	51.36	51.68%
1	1.344%	3.124	51.69	51.6%
2	1.346%	3.148	52.03	51.72%
3	1.344%	3.122	51.68	52.97%
4	1.349%	3.158	52.25	52.4%
5	1.35%	3.159	52.32	52.0%

6	1.35%	3.172	52.8	51.56%
7	1.343%	3.116	51.39	51.88%
8	1.35%	3.157	52.79	52.44%
9	1.348%	3.144	51.71	52.57%
Mean	1.347%	3.143	52.0	52.08%
Mean Rank	1	1	1	1
Sum rank	4			
price, [160]				
0	1.349%	3.16	52.52	50.91%
1	1.354%	3.212	53.89	51.07%
2	1.345%	3.125	51.93	53.17%
3	1.346%	3.132	52.23	52.2%
4	1.35%	3.171	53.12	50.95%
5	1.347%	3.144	52.11	52.44%
6	1.351%	3.167	52.73	52.24%
7	1.349%	3.142	52.47	52.53%
8	1.347%	3.14	52.09	52.0%
9	1.344%	3.149	52.41	50.79%
Mean	1.348%	3.154	52.55	51.83%
Mean Rank	2	2	2	2
Sum rank	8			
price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.351%	3.162	53.25	51.47%
1	1.349%	3.169	52.91	52.36%
2	1.352%	3.217	54.62	52.08%
3	1.356%	3.18	52.96	51.07%
4	1.362%	3.156	52.27	52.12%
5	1.352%	3.152	52.33	51.11%
6	1.356%	3.213	54.68	52.36%
7	1.351%	3.143	53.09	51.6%
8	1.352%	3.138	51.58	52.2%

9	1.359%	3.183	53.17	51.52%
Mean	1.354%	3.171	53.09	51.79%
Mean Rank	3	3	3	3
Sum rank	12			

Table 63: Full table: Optimising model related to each feature subset 10 times

B.5 Predicting price by optimising on price and direction

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, open, high, low, volume, direction, price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, prev_change_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_price_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_trendscore_0, prev_change_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_price_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, prev_trendscore_1, [160]				
0	1.356%	3.136	52.31	51.03%
1	1.352%	3.119	51.95	51.23%
2	1.35%	3.122	52.44	51.72%
Mean	1.353%	3.126	52.23	51.33%

Mean Rank	2	1	2	1
Sum rank	6			
change, open, high, low, volume, direction, price, prev_change_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_price_0, prev_change_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_price_1, [160]				
0	1.352%	3.135	51.83	49.98%
1	1.351%	3.131	51.95	51.52%
2	1.359%	3.139	51.61	51.35%
Mean	1.354%	3.135	51.8	50.95%
Mean Rank	3	2	1	2
Sum rank	8			
change, positive, prev_change_0, prev_positive_0, prev_change_1, prev_positive_1, [160]				
0	1.353%	3.167	52.78	48.12%
1	1.351%	3.137	52.52	50.87%
2	1.348%	3.123	52.22	50.59%
Mean	1.35%	3.142	52.51	49.86%
Mean Rank	1	3	3	5
Sum rank	12			

change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, prev_change_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_change_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, [160]				
0	1.363%	3.178	53.65	50.1%
1	1.361%	3.162	53.4	50.71%
2	1.365%	3.161	53.42	51.03%
Mean	1.363%	3.167	53.49	50.61%
Mean Rank	4	4	4	3
Sum rank	15			
change, trendscore, prev_change_0, prev_trendscore_0, prev_change_1, prev_trendscore_1, [160]				
0	1.39%	3.211	54.84	49.9%
1	1.378%	3.159	52.81	49.74%
2	1.39%	3.183	53.33	50.1%
Mean	1.386%	3.185	53.66	49.91%
Mean Rank	5	5	5	4
Sum rank	19			

Table 64: Full table: Predicting price and direction.

B.6 Additional experiment: Predicting price change

Seed	**MAPE**	**MAE**	**MSE**	**DA**

price, open, high, low, volume, direction, change, [160]				
0	1.371%	3.135	52.59	51.72%
1	1.348%	3.104	51.74	53.09%
2	1.348%	3.146	52.82	51.03%
Mean	1.356%	3.128	52.38	51.95%
Mean Rank	1	1	1	2
Sum rank	5			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	1.37%	3.228	55.46	50.91%
1	1.356%	3.125	52.89	53.01%
2	1.348%	3.129	52.57	52.08%
Mean	1.358%	3.161	53.64	52.0%
Mean Rank	2	2	2	1
Sum rank	7			
price, trendscore, [160]				
0	1.972%	4.328	98.53	51.52%
1	1.901%	4.676	122.0	51.15%
2	1.898%	4.489	108.2	51.47%
Mean	1.923%	4.498	109.6	51.38%
Mean Rank	3	3	3	5
Sum rank	14			
price, [160]				
0	1.931%	4.975	146.3	51.11%
1	1.951%	4.242	92.38	51.27%
2	1.9%	4.641	120.9	51.84%
Mean	1.927%	4.62	119.9	51.41%
Mean Rank	5	4	4	4

Sum rank	17			
price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.879%	4.445	106.8	51.68%
1	2.006%	5.245	159.3	51.8%
2	1.889%	4.485	109.0	50.79%
Mean	1.925%	4.725	125.1	51.42%
Mean Rank	4	5	5	3
Sum rank	17			

Table 65: Full table: Price change prediction without context module.

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, open, high, low, volume, direction, change, [160]				
0	1.371%	3.173	53.57	51.96%
1	1.367%	3.218	55.34	50.91%
2	1.386%	3.15	52.9	51.03%
Mean	1.374%	3.18	53.93	51.3%
Mean Rank	1	1	2	3
Sum rank	7			
change, [160]				
0	1.357%	3.156	52.85	49.41%
1	1.42%	3.211	54.27	49.21%
2	1.381%	3.2	54.32	49.33%
Mean	1.386%	3.189	53.81	49.32%
Mean Rank	2	2	1	5
Sum rank	10			
price, open, high, low, [160]				
0	2.003%	5.517	180.0	52.32%
1	1.99%	4.859	134.5	51.03%

2	1.876%	4.177	90.33	51.8%
Mean	1.957%	4.851	135.0	51.72%
Mean Rank	4	3	3	1
Sum rank	11			
volume, [160]				
0	1.992%	5.253	166.1	50.34%
1	1.956%	5.068	156.0	50.71%
2	1.904%	4.89	147.0	51.68%
Mean	1.951%	5.07	156.4	50.91%
Mean Rank	3	4	4	4
Sum rank	15			
direction, [160]				
0	2.676%	5.835	157.3	51.39%
1	7.489%	12.24	378.8	50.91%
2	2.784%	8.137	343.4	52.77%
Mean	4.316%	8.737	293.1	51.69%
Mean Rank	5	5	5	2
Sum rank	17			

Table 66: Full table: Predicting price change, analysing different subsets of the trading data

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, open, high, low, volume, direction, price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	1.352%	3.12	52.28	52.32%
1	1.383%	3.156	52.14	50.46%
2	1.357%	3.135	52.61	52.77%
Mean	1.364%	3.137	52.34	51.85%
Mean Rank	3	1	1	1
Sum rank	6			

change, open, high, low, volume, direction, price, [160]				
0	1.354%	3.144	52.7	50.95%
1	1.348%	3.123	52.1	52.85%
2	1.363%	3.151	53.28	51.43%
Mean	1.355%	3.139	52.69	51.74%
Mean Rank	1	2	2	2
Sum rank	7			
change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.363%	3.163	53.55	49.82%
1	1.36%	3.17	53.36	51.11%
2	1.361%	3.168	53.64	50.83%
Mean	1.361%	3.167	53.52	50.59%
Mean Rank	2	3	3	3
Sum rank	11			
change, trendscore, [160]				
0	1.432%	3.196	54.24	48.85%
1	1.417%	3.207	55.54	50.3%
2	1.4%	3.209	55.86	49.78%
Mean	1.416%	3.204	55.21	49.64%
Mean Rank	4	4	5	4
Sum rank	17			
change, [160]				
0	1.481%	3.214	54.15	48.73%
1	1.511%	3.209	54.23	49.94%
2	1.409%	3.211	54.65	49.66%
Mean	1.467%	3.211	54.34	49.44%
Mean Rank	5	5	4	5
Sum rank	19			

Table 67: Full table: Price change prediction without context module.

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, open, high, low, price, [160]				
0	1.377%	3.172	52.92	51.07%
1	1.463%	3.29	57.43	48.08%
2	1.461%	3.137	52.61	51.03%
Mean	1.434%	3.2	54.32	50.06%
Mean Rank	3	2	1	1
Sum rank	7			
change, volume, [160]				
0	1.374%	3.194	54.33	50.71%
1	1.411%	3.213	54.35	48.65%
2	1.401%	3.181	54.41	50.26%
Mean	1.395%	3.196	54.36	49.87%
Mean Rank	2	1	3	2
Sum rank	8			
change, direction, [160]				
0	1.382%	3.218	54.47	49.54%
1	1.394%	3.197	54.34	49.9%
2	1.391%	3.205	54.25	49.98%
Mean	1.389%	3.207	54.35	49.8%
Mean Rank	1	3	2	3
Sum rank	9			

Table 68: Full table: Splitting up the trading features in order to identify patterns.

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.345%	3.123	51.92	51.52%

1	1.346%	3.126	51.92	51.96%
2	1.349%	3.133	52.41	51.96%
Mean	1.347%	3.127	52.08	51.81%
Mean Rank	1	1	1	1
Sum rank	4			
change, open, high, low, volume, direction, price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	1.366%	3.17	53.35	50.18%
1	1.361%	3.126	52.06	52.53%
2	1.371%	3.165	53.36	50.34%
Mean	1.366%	3.153	52.92	51.02%
Mean Rank	3	2	2	2
Sum rank	9			
change, open, high, low, volume, direction, price, [160]				
0	1.351%	3.183	53.85	49.45%
1	1.387%	3.165	52.76	50.87%
2	1.353%	3.146	53.28	51.56%
Mean	1.364%	3.165	53.3	50.63%
Mean Rank	2	3	3	3
Sum rank	11			
change, [160]				
0	1.417%	3.165	53.12	49.94%
1	1.413%	3.169	53.49	50.51%
2	1.407%	3.174	53.37	50.55%
Mean	1.412%	3.17	53.33	50.33%
Mean Rank	5	4	4	5
Sum rank	18			
change, trendscore, [160]				
0	1.421%	3.177	53.86	50.51%

1	1.39%	3.166	52.81	49.7%
2	1.392%	3.21	56.77	51.31%
Mean	1.401%	3.184	54.48	50.51%
Mean Rank	4	5	5	4
Sum rank	18			

Table 69: Full table: Price change prediction with context module.

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, [160]				
0	1.369%	3.184	53.89	49.21%
1	1.366%	3.187	53.95	49.09%
2	1.37%	3.158	53.28	49.49%
Mean	1.368%	3.176	53.71	49.27%
Mean Rank	1	1	1	2
Sum rank	5			
price, [160]				
0	1.952%	4.92	139.0	51.72%
1	1.919%	4.371	101.9	51.27%
2	1.828%	4.331	103.2	50.71%
Mean	1.9%	4.541	114.7	51.23%
Mean Rank	2	2	2	1
Sum rank	7			

Table 70: Full table: Comparing using only price to using only price change when predicting the next change.

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, negative, neutral, [160]				
0	1.344%	3.116	52.24	50.95%
1	1.348%	3.134	52.28	51.84%

2	1.345%	3.118	51.96	51.19%
Mean	1.346%	3.123	52.16	51.33%
Mean Rank	1	1	1	2
Sum rank	5			
change, positive_prop, negative_prop, neutral_prop, [160]				
0	1.387%	3.153	53.1	50.51%
1	1.353%	3.14	52.74	51.6%
2	1.352%	3.142	53.05	51.68%
Mean	1.364%	3.145	52.96	51.26%
Mean Rank	2	2	2	3
Sum rank	9			
positive, negative, neutral, [160]				
0	1.451%	3.393	61.15	50.26%
1	1.377%	3.145	52.18	52.0%
2	1.392%	3.177	52.28	52.32%
Mean	1.407%	3.238	55.2	51.53%
Mean Rank	3	3	4	1
Sum rank	11			
positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.443%	3.279	55.66	50.38%
1	1.4%	3.158	52.92	51.6%
2	1.412%	3.286	56.95	51.56%
Mean	1.418%	3.241	55.17	51.18%
Mean Rank	4	4	3	4
Sum rank	15			
positive_prop, negative_prop, neutral_prop, [160]				
0	1.56%	3.426	64.83	50.59%
1	1.502%	3.34	58.89	50.95%
2	1.396%	3.249	56.33	51.47%

Mean	1.486%	3.338	60.02	51.0%
Mean Rank	5	5	5	5
Sum rank	20			

Table 71: Full table: Identifying how much each feature subset contributes to the configuration that yielded the best results in Table 28.

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, negative, [160]				
0	1.35%	3.134	52.4	51.92%
1	1.343%	3.115	51.96	51.92%
2	1.345%	3.128	52.05	51.72%
Mean	1.346%	3.126	52.13	51.85%
Mean Rank	1	1	1	1
Sum rank	4			
change, positive, [160]				
0	1.349%	3.132	52.29	50.95%
1	1.347%	3.126	52.13	51.96%
2	1.346%	3.127	51.99	51.11%
Mean	1.347%	3.129	52.14	51.34%
Mean Rank	2	2	2	3
Sum rank	9			
change, positive, neutral, [160]				
0	1.347%	3.131	52.2	50.99%
1	1.347%	3.127	52.52	51.76%
2	1.354%	3.135	52.67	52.08%
Mean	1.349%	3.131	52.46	51.61%
Mean Rank	3	3	3	2
Sum rank	11			
change, negative, [160]				
0	1.349%	3.135	51.9	51.39%
1	1.364%	3.152	53.63	50.51%

2	1.369%	3.144	53.33	50.87%
Mean	1.361%	3.144	52.95	50.92%
Mean Rank	4	4	5	4
Sum rank	17			
change, neutral, [160]				
0	1.41%	3.17	53.39	50.14%
1	1.434%	3.161	52.5	49.58%
2	1.35%	3.135	52.21	51.76%
Mean	1.398%	3.155	52.7	50.49%
Mean Rank	5	5	4	5
Sum rank	19			

Table 72: Full table: Identifying how much each feature subset contributes to the configuration that yielded the best results in Table 29.

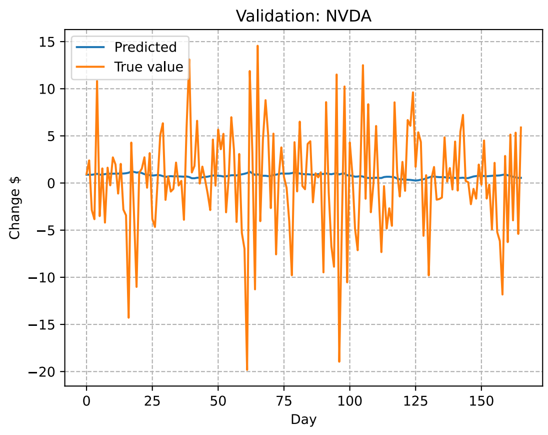
Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, trendscore, [160]				
0	1.356%	3.151	52.75	51.43%
1	1.349%	3.127	52.13	52.16%
2	1.349%	3.124	52.12	50.75%
Mean	1.351%	3.134	52.34	51.45%
Mean Rank	1	1	1	1
Sum rank	4			

Table 73: Full table: Introducing trendscore.

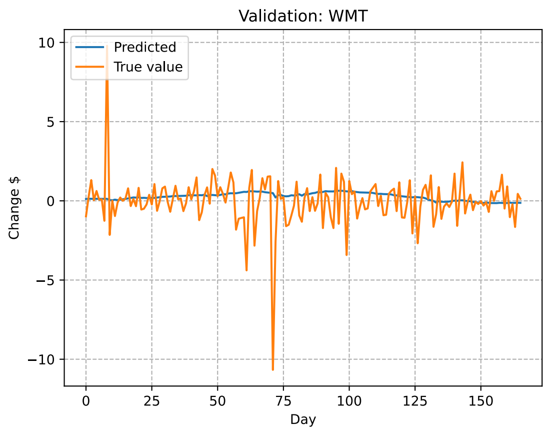
Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, volume, direction, [160]				
	1.345%	3.122	52.52	51.8%
	1.355%	3.155	53.08	50.55%

	1.347%	3.126	52.33	50.95%
Mean	1.349%	3.134	52.64	51.1%
Mean Rank	1	1	1	1
Sum rank	4			
change, positive, open, high, low, price, volume, direction, [160]				
	1.362%	3.249	56.45	50.91%
	1.388%	3.204	54.69	50.63%
	1.392%	3.247	56.22	49.58%
Mean	1.381%	3.233	55.79	50.37%
Mean Rank	2	2	2	2
Sum rank	8			
change, positive, open, high, low, price, [160]				
	1.434%	3.195	53.95	49.45%
	1.374%	3.264	56.17	49.01%
	1.37%	3.282	57.41	49.74%
Mean	1.393%	3.247	55.84	49.4%
Mean Rank	3	3	3	3
Sum rank	12			

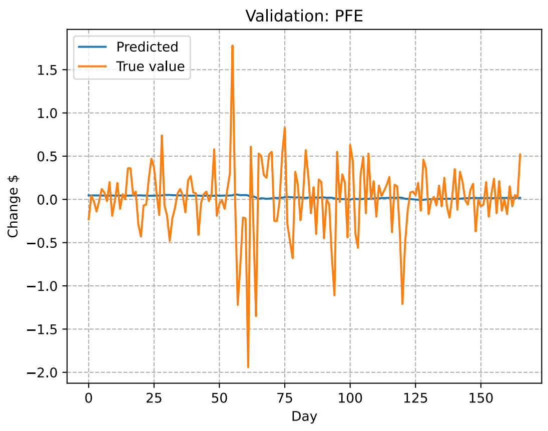
Table 74: Full table: Introducing trading data



(a) Results for the NVDA stock, using "change" and trading data.

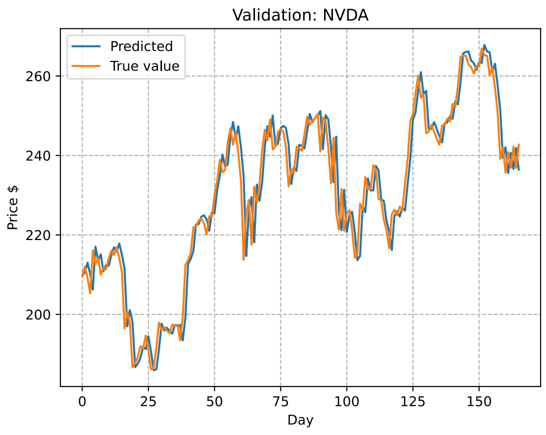


(b) Results for the NFLX stock, using "change" and "trendscore".

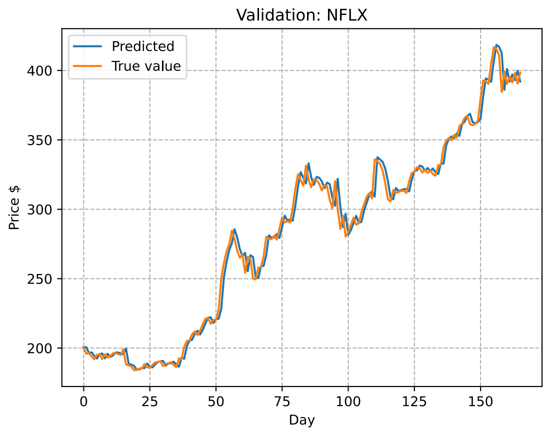


(c) Results for the KO stock, using "change" only.

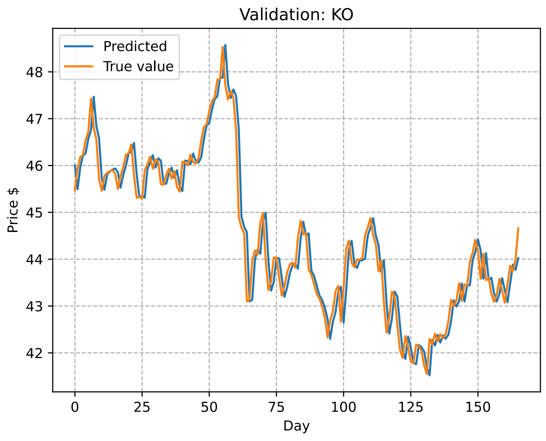
Figure 45: Examples showing the mean of the predictions to be slightly above 0.



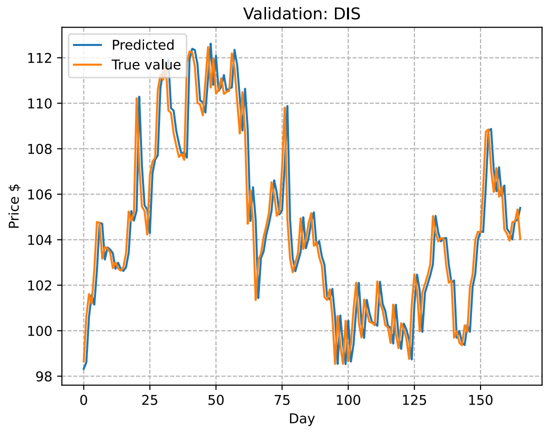
(a) Results for the NVDA stock, using "change" and "negative".



(b) Results for the NFLX stock, using "change", "neutral" and "positive".



(c) Results for the KO stock, using "change", "positive" and "negative".



(d) Results for the DIS stock, using "change" and "positive".

Figure 46: Examples showing the added price changes making little difference in prediction. Trained on sentiment divided into smaller subsets.

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, open_close_change, high_close_change, low_close_change, [160]				
0	1.359%	3.188	56.8	51.15%

1	1.362%	3.172	55.19	52.12%
2	1.359%	3.178	55.55	50.22%
Mean	1.36%	3.179	55.85	51.16%
Mean Rank	1	2	3	1
Sum rank	7			
change, trend-score_change, [160]				
0	1.375%	3.157	53.52	50.14%
1	1.403%	3.172	53.43	49.9%
2	1.348%	3.136	51.98	51.03%
Mean	1.375%	3.155	52.97	50.36%
Mean Rank	2	1	1	3
Sum rank	7			
change, open, high, low, volume, direction, price, open_close_change, high_close_change, low_close_change, [160]				
0	1.427%	3.223	55.85	49.41%
1	1.352%	3.136	53.44	51.84%
2	1.362%	3.199	54.32	50.67%
Mean	1.38%	3.186	54.54	50.64%
Mean Rank	3	3	2	2
Sum rank	10			

Table 75: Full table: Using price differences between open, low and high, and close price or trendscore change.

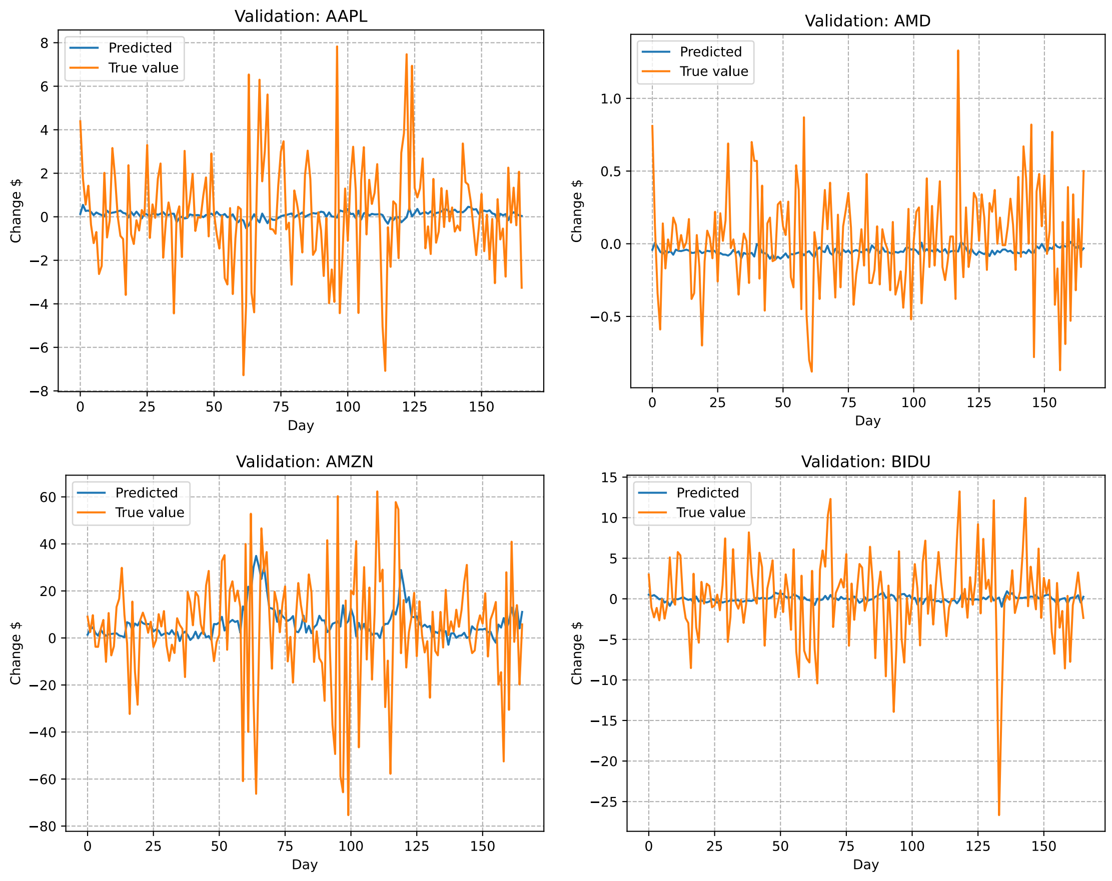


Figure 47: Using price differences between open, low and high, and close price or trend-score change

B.7 Additional experiment: Predicting next price change using several time steps

Seed	MAPE	MAE	MSE	DA

change, open, high, low, volume, direction, price, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, prev_change_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_price_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_trendscore_0, prev_change_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_price_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, prev_trendscore_1, [160]				
0	1.356%	3.136	52.31	51.03%
1	1.352%	3.119	51.95	51.23%
2	1.35%	3.122	52.44	51.72%
Mean	1.353%	3.126	52.23	51.33%
Mean Rank	2	1	2	1
Sum rank	6			
change, open, high, low, volume, direction, price, prev_change_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_price_0, prev_change_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_price_1, [160]				

0	1.352%	3.135	51.83	49.98%
1	1.351%	3.131	51.95	51.52%
2	1.359%	3.139	51.61	51.35%
Mean	1.354%	3.135	51.8	50.95%
Mean Rank	3	2	1	2
Sum rank	8			
change, positive, prev_change_0, prev_positive_0, prev_change_1, prev_positive_1, [160]				
0	1.353%	3.167	52.78	48.12%
1	1.351%	3.137	52.52	50.87%
2	1.348%	3.123	52.22	50.59%
Mean	1.35%	3.142	52.51	49.86%
Mean Rank	1	3	3	5
Sum rank	12			
change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, prev_change_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_change_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, [160]				
0	1.363%	3.178	53.65	50.1%
1	1.361%	3.162	53.4	50.71%
2	1.365%	3.161	53.42	51.03%
Mean	1.363%	3.167	53.49	50.61%
Mean Rank	4	4	4	3
Sum rank	15			

change, trendscore, prev_change_0, prev_trendscore_0, prev_change_1, prev_trendscore_1, [160]				
0	1.39%	3.211	54.84	49.9%
1	1.378%	3.159	52.81	49.74%
2	1.39%	3.183	53.33	50.1%
Mean	1.386%	3.185	53.66	49.91%
Mean Rank	5	5	5	4
Sum rank	19			

Table 76: Full table: Using several time steps

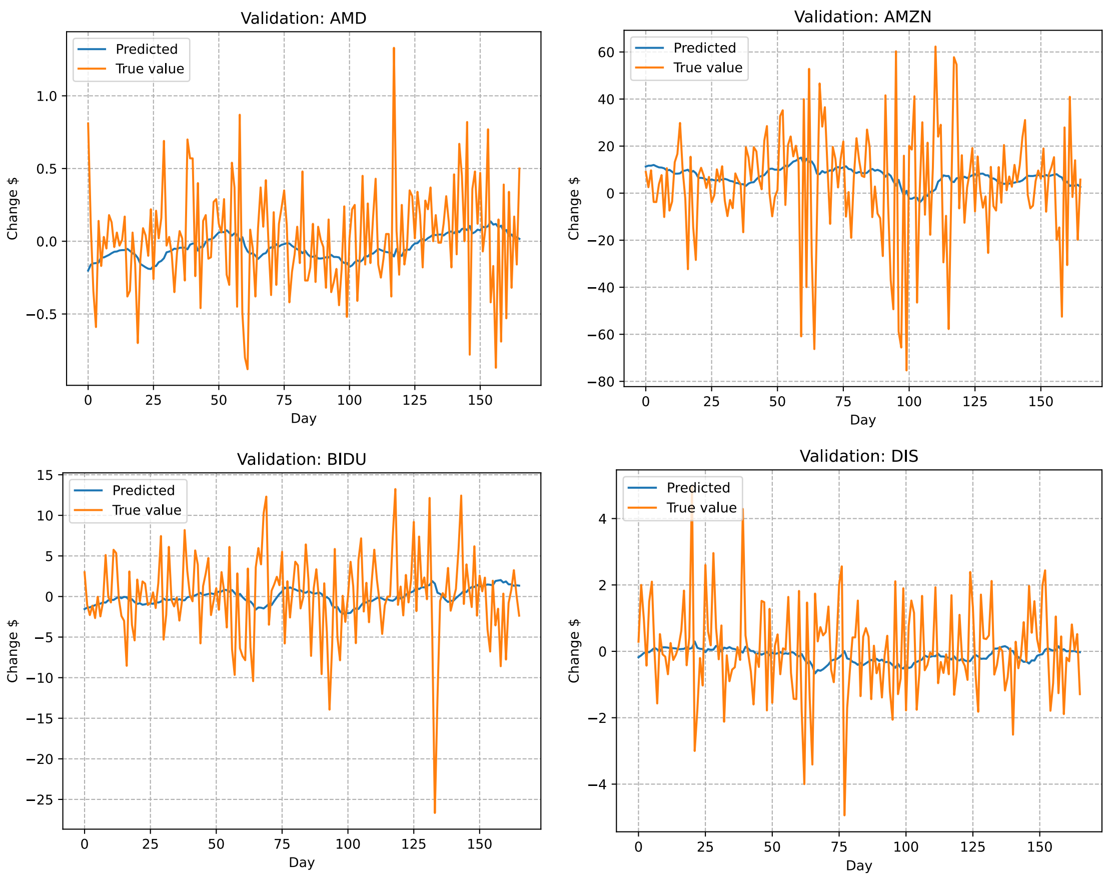


Figure 48: Predicting next change when using change[0-2] and trendscore[0-2]

B.8 Additional experiment: Predicting price change for one stock at a time

Seed	**MAPE**	**MAE**	**MSE**	**DA**
trendscore, [160]				
0	1.058%	1.836	6.024	56.36%
1	1.005%	1.743	5.743	61.21%
2	1.01%	1.752	5.729	54.55%
Mean	1.025%	1.777	5.832	57.37%
Mean Rank	4	4	1	1
Sum rank	10			
change, [160]				
0	1.019%	1.766	5.816	52.12%
1	1.027%	1.779	5.868	44.85%
2	1.023%	1.772	5.845	50.3%
Mean	1.023%	1.772	5.843	49.09%
Mean Rank	1	1	2	7
Sum rank	11			

price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, prev_price_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_change_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_trendscore_0, prev_price_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_change_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, prev_trendscore_1, [160]				
0	1.014%	1.756	5.808	56.97%
1	1.025%	1.775	5.859	51.52%
2	1.033%	1.791	5.961	53.94%
Mean	1.024%	1.774	5.876	54.14%
Mean Rank	3	2	4	2
Sum rank	11			
change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.021%	1.77	5.846	50.91%
1	1.025%	1.777	5.861	48.48%
2	1.025%	1.777	5.867	50.3%
Mean	1.024%	1.775	5.858	49.9%
Mean Rank	2	3	3	6

Sum rank	14			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	1.032%	1.79	5.894	51.52%
1	1.021%	1.77	5.85	53.94%
2	1.039%	1.801	5.934	47.88%
Mean	1.031%	1.787	5.893	51.11%
Mean Rank	5	5	5	5
Sum rank	20			
change, trendscore, [160]				
0	1.035%	1.794	5.907	49.7%
1	1.025%	1.777	5.839	50.91%
2	1.049%	1.818	6.027	58.79%
Mean	1.036%	1.796	5.924	53.13%
Mean Rank	6	6	6	3
Sum rank	21			
change, open, high, low, volume, direction, price, [160]				
0	1.067%	1.85	6.314	52.12%
1	1.039%	1.8	6.008	50.3%
2	1.026%	1.778	5.951	51.52%
Mean	1.044%	1.809	6.091	51.31%
Mean Rank	7	7	7	4
Sum rank	25			

Table 77: Full table: Feature search on the AAPL stock

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, [160]				
0	1.288%	2.292	10.59	56.36%

1	1.287%	2.292	10.58	57.58%
2	1.288%	2.293	10.59	57.58%
Mean	1.288%	2.292	10.58	57.17%
Mean Rank	1	1	3	2
Sum rank	7			
change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.291%	2.298	10.6	61.21%
1	1.287%	2.29	10.56	64.24%
2	1.287%	2.291	10.56	60.0%
Mean	1.288%	2.293	10.57	61.82%
Mean Rank	2	2	2	1
Sum rank	7			
change, trendscore, [160]				
0	1.299%	2.308	10.46	53.94%
1	1.291%	2.295	10.43	53.33%
2	1.314%	2.334	10.9	50.3%
Mean	1.301%	2.312	10.6	52.53%
Mean Rank	4	4	4	5
Sum rank	17			
price, open, high, low, volume, direction, change, [160]				
0	1.299%	2.311	10.79	52.73%
1	1.297%	2.307	10.85	53.33%
2	1.299%	2.311	10.77	53.33%
Mean	1.298%	2.31	10.8	53.13%
Mean Rank	3	3	7	4
Sum rank	17			
trendscore, [160]				
0	1.302%	2.318	10.38	48.48%
1	1.305%	2.323	10.56	52.12%
2	1.304%	2.321	10.48	50.3%

Mean	1.304%	2.321	10.47	50.3%
Mean Rank	5	5	1	7
Sum rank	18			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	1.305%	2.322	10.71	51.52%
1	1.304%	2.319	10.79	56.36%
2	1.313%	2.337	10.76	53.33%
Mean	1.307%	2.326	10.75	53.74%
Mean Rank	6	6	5	3
Sum rank	20			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, prev_price_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_change_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_trendscore_0, prev_price_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_change_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, prev_trendscore_1, [160]				
0	1.321%	2.35	10.71	51.52%

1	1.316%	2.341	10.61	50.91%
2	1.313%	2.336	10.98	49.7%
Mean	1.317%	2.342	10.77	50.71%
Mean Rank	7	7	6	6
Sum rank	26			

Table 78: Full table: Feature search on the FB stock

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, open, high, low, volume, direction, change, [160]				
0	0.9412%	1.735	5.925	55.15%
1	0.9413%	1.735	5.923	53.94%
2	0.9408%	1.734	5.922	53.94%
Mean	0.9411%	1.735	5.924	54.34%
Mean Rank	1	1	2	3
Sum rank	7			

price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, prev_price_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_change_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_trendscore_0, prev_price_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_change_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, prev_trendscore_1, [160]				
0	0.9419%	1.736	5.913	55.76%
1	0.9384%	1.73	5.882	57.58%
2	0.9436%	1.738	5.965	53.94%
Mean	0.9413%	1.735	5.92	55.76%
Mean Rank	2	2	1	2
Sum rank	7			
change, [160]				
0	0.9477%	1.747	5.971	53.94%
1	0.9414%	1.735	5.907	54.55%
2	0.9397%	1.732	5.904	53.33%
Mean	0.9429%	1.738	5.927	53.94%
Mean Rank	3	3	3	4
Sum rank	13			

price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	0.9423%	1.737	5.951	59.39%
1	0.9445%	1.741	5.98	57.58%
2	0.9453%	1.742	5.985	60.0%
Mean	0.944%	1.74	5.972	58.99%
Mean Rank	4	4	4	1
Sum rank	13			
change, trendscore, [160]				
0	0.9455%	1.743	5.977	50.3%
1	0.9444%	1.741	5.964	49.7%
2	0.9464%	1.744	5.994	52.12%
Mean	0.9454%	1.743	5.978	50.71%
Mean Rank	5	5	5	5
Sum rank	20			
change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	0.9523%	1.759	6.054	48.48%
1	0.9506%	1.752	6.022	53.94%
2	0.9591%	1.77	6.082	46.06%
Mean	0.954%	1.761	6.053	49.49%
Mean Rank	6	6	6	6
Sum rank	24			
trendscore, [160]				
0	0.9483%	1.748	5.961	50.91%
1	0.9396%	1.733	5.881	56.36%
2	0.9907%	1.82	6.403	44.85%
Mean	0.9595%	1.767	6.082	50.71%
Mean Rank	7	7	7	5

Sum rank	26

Table 79: Full table: Feature search on the HD stock

B.9 Summary and evaluation on the test set

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, trendscore, [160]				
0	1.356%	3.181	54.0	51.56%
1	1.348%	3.125	51.32	52.2%
2	1.358%	3.157	52.24	52.08%
Mean	1.354%	3.154	52.52	51.95%
Mean Rank	1	1	1	2
Sum rank	5			
price, positive, [160]				
0	1.352%	3.149	53.56	52.0%
1	1.357%	3.154	52.59	52.16%
2	1.359%	3.163	52.35	53.21%
Mean	1.356%	3.155	52.84	52.46%
Mean Rank	2	2	2	1
Sum rank	7			
price, [160]				
0	1.355%	3.202	53.43	51.88%
1	1.348%	3.14	52.49	51.92%
2	1.369%	3.332	58.2	50.75%
Mean	1.358%	3.225	54.71	51.52%
Mean Rank	3	3	3	3
Sum rank	12			
price, open, high, low, volume, direction, change, [160]				
0	1.365%	3.257	56.57	51.03%
1	1.364%	3.257	58.38	50.71%

2	1.367%	3.268	60.15	52.4%
Mean	1.365%	3.26	58.37	51.38%
Mean Rank	4	4	4	4
Sum rank	16			

Table 80: Full table: Predicting **next price** without context on the **validation** set

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, positive, [160]				
0	1.691%	4.627	149.5	51.64%
1	1.693%	4.528	148.4	50.87%
2	1.68%	4.51	141.5	51.64%
Mean	1.688%	4.555	146.5	51.38%
Mean Rank	3	1	1	3
Sum rank	8			
price, trendscore, [160]				
0	1.684%	4.571	149.9	52.16%
1	1.695%	4.578	152.1	51.72%
2	1.684%	4.546	147.4	52.16%
Mean	1.688%	4.565	149.8	52.01%
Mean Rank	2	2	3	2
Sum rank	9			
price, open, high, low, volume, direction, change, [160]				
0	1.679%	4.584	148.9	53.98%
1	1.695%	4.62	147.7	53.9%
2	1.71%	4.656	150.9	53.09%
Mean	1.695%	4.62	149.2	53.66%
Mean Rank	4	3	2	1
Sum rank	10			
price, [160]				
0	1.669%	4.546	146.1	50.34%

1	1.669%	4.469	143.5	51.52%
2	1.709%	4.938	169.0	50.87%
Mean	1.682%	4.651	152.8	50.91%
Mean Rank	1	4	4	4
Sum rank	13			

Table 81: Full table: Predicting **next price** without context on the **test** set

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, [160]				
0	1.355%	3.158	52.3	50.63%
1	1.348%	3.13	52.56	51.07%
2	1.352%	3.131	52.68	51.8%
Mean	1.352%	3.139	52.52	51.16%
Mean Rank	2	1	2	2
Sum rank	7			
change, positive, [160]				
0	1.346%	3.137	52.21	49.29%
1	1.348%	3.15	52.33	50.59%
2	1.344%	3.131	51.93	51.15%
Mean	1.346%	3.14	52.16	50.34%
Mean Rank	1	2	1	4
Sum rank	8			
change, trendscore, [160]				
0	1.407%	3.174	54.34	50.99%
1	1.348%	3.134	52.4	52.0%
2	1.352%	3.116	51.89	52.16%
Mean	1.369%	3.141	52.88	51.72%
Mean Rank	4	3	3	1
Sum rank	11			
price, open, high, low, volume, direction, change, [160]				

	MAPE	MAE	MSE	DA
0	1.363%	3.2	56.7	50.1%
1	1.371%	3.164	54.0	50.42%
2	1.341%	3.099	52.74	51.92%
Mean	1.359%	3.154	54.48	50.81%
Mean Rank	3	4	4	3
Sum rank	14			

Table 82: Full table: Predicting **next price change** without context on the **validation** set

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, [160]				
0	1.665%	4.449	144.3	50.34%
1	1.666%	4.441	143.9	50.87%
2	1.661%	4.438	143.7	51.68%
Mean	1.664%	4.443	144.0	50.96%
Mean Rank	1	1	1	3
Sum rank	6			
change, [160]				
0	1.679%	4.451	143.6	50.51%
1	1.666%	4.447	145.7	50.79%
2	1.671%	4.456	145.5	50.63%
Mean	1.672%	4.451	145.0	50.64%
Mean Rank	2	2	2	4
Sum rank	10			
change, trendscore, [160]				
0	1.71%	4.531	150.1	51.35%
1	1.669%	4.54	151.1	51.6%
2	1.667%	4.441	144.5	50.26%
Mean	1.682%	4.504	148.6	51.07%
Mean Rank	3	3	3	2
Sum rank	11			

price, open, high, low, volume, direction, change, [160]				
0	1.723%	4.762	170.2	50.87%
1	1.677%	4.445	143.4	51.39%
2	1.706%	4.519	151.2	51.84%
Mean	1.702%	4.575	154.9	51.37%
Mean Rank	4	4	4	1
Sum rank	13			

Table 83: Full table: Predicting **next price change** without context on the **test** set

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, trendscore, [160]				
0	1.347%	3.143	51.91	52.24%
1	1.347%	3.141	51.67	52.04%
2	1.346%	3.143	51.79	52.53%
Mean	1.347%	3.142	51.79	52.27%
Mean Rank	1	1	1	2
Sum rank	5			
price, positive, [160]				
0	1.355%	3.138	53.66	52.93%
1	1.349%	3.153	52.77	52.24%
2	1.347%	3.154	52.27	52.16%
Mean	1.35%	3.149	52.9	52.44%
Mean Rank	3	2	3	1
Sum rank	9			
price, [160]				
0	1.347%	3.166	52.8	51.84%
1	1.348%	3.163	52.79	51.31%
2	1.347%	3.158	52.48	51.72%
Mean	1.347%	3.162	52.69	51.62%
Mean Rank	2	3	2	3

Sum rank	10			
price, open, high, low, volume, direction, change, [160]				
0	1.372%	3.197	55.0	50.67%
1	1.367%	3.233	58.21	50.06%
2	1.369%	3.225	58.13	50.14%
Mean	1.369%	3.218	57.12	50.29%
Mean Rank	4	4	4	4
Sum rank	16			

Table 84: Full table: Predicting **next price** using context on the **validation** set

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, [160]				
0	1.662%	4.45	143.0	51.68%
1	1.662%	4.468	143.1	50.38%
2	1.662%	4.451	143.3	50.18%
Mean	1.662%	4.456	143.1	50.75%
Mean Rank	1	1	1	4
Sum rank	7			
price, trendscore, [160]				
0	1.665%	4.473	145.3	50.91%
1	1.663%	4.449	141.9	52.36%
2	1.664%	4.457	142.4	52.77%
Mean	1.664%	4.459	143.2	52.01%
Mean Rank	2	2	2	2
Sum rank	8			
price, positive, [160]				
0	1.683%	4.523	148.2	51.72%
1	1.673%	4.479	141.2	51.52%
2	1.668%	4.461	142.5	50.79%
Mean	1.675%	4.488	144.0	51.34%

Mean Rank	3	3	3	3
Sum rank	12			
price, open, high, low, volume, direction, change, [160]				
0	1.694%	4.621	156.9	52.53%
1	1.687%	4.521	151.3	53.45%
2	1.704%	4.671	161.2	53.49%
Mean	1.695%	4.604	156.5	53.16%
Mean Rank	4	4	4	1
Sum rank	13			

Table 85: Full table: Predicting **next price** using context on the **test** set

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, [160]				
0	1.346%	3.122	52.58	52.16%
1	1.343%	3.112	51.84	51.68%
2	1.342%	3.121	51.51	53.01%
Mean	1.344%	3.118	51.98	52.28%
Mean Rank	1	1	1	1
Sum rank	4			
change, trendscore, [160]				
0	1.353%	3.132	51.95	51.15%
1	1.391%	3.177	54.96	50.38%
2	1.35%	3.126	51.61	51.88%
Mean	1.365%	3.145	52.84	51.14%
Mean Rank	3	2	3	2
Sum rank	10			
change, [160]				
0	1.349%	3.128	52.41	51.88%
1	1.415%	3.173	53.3	50.42%
2	1.353%	3.142	51.89	50.99%

Mean	1.372%	3.148	52.53	51.1%
Mean Rank	4	3	2	3
Sum rank	12			
price, open, high, low, volume, direction, change, [160]				
0	1.36%	3.204	55.08	48.65%
1	1.349%	3.131	52.57	51.03%
2	1.377%	3.147	52.92	51.64%
Mean	1.362%	3.161	53.52	50.44%
Mean Rank	2	4	4	4
Sum rank	14			

Table 86: Full table: Predicting **next price change** using context on the **validation** set

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, [160]				
0	1.667%	4.46	145.5	51.27%
1	1.665%	4.448	144.6	51.27%
2	1.668%	4.452	144.0	50.3%
Mean	1.667%	4.453	144.7	50.95%
Mean Rank	1	1	1	1
Sum rank	4			
change, [160]				
0	1.668%	4.458	145.7	50.22%
1	1.734%	4.499	145.7	50.67%
2	1.683%	4.453	144.1	50.18%
Mean	1.695%	4.47	145.2	50.36%
Mean Rank	3	2	2	4
Sum rank	11			
change, trendscore, [160]				
0	1.68%	4.464	144.6	49.78%
1	1.707%	4.536	150.3	50.67%

2	1.672%	4.465	145.0	50.71%
Mean	1.686%	4.488	146.6	50.38%
Mean Rank	2	3	4	3
Sum rank	12			
price, open, high, low, volume, direction, change, [160]				
0	1.68%	4.526	146.6	49.66%
1	1.68%	4.504	149.0	50.14%
2	1.729%	4.447	144.1	51.84%
Mean	1.696%	4.492	146.6	50.55%
Mean Rank	4	4	3	2
Sum rank	13			

Table 87: Full table: Predicting **next price change** using context on the **test** set

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.49%	2.806	15.57	53.33%
1	1.487%	2.799	15.5	52.12%
2	1.49%	2.805	15.56	51.52%
Mean	1.489%	2.803	15.54	52.32%
Mean Rank	1	1	1	2
Sum rank	5			
change, [160]				
0	1.491%	2.81	15.66	43.64%
1	1.494%	2.812	15.59	50.91%
2	1.49%	2.806	15.55	49.7%
Mean	1.492%	2.809	15.6	48.08%
Mean Rank	2	2	2	6
Sum rank	12			

price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	1.496%	2.821	15.72	48.48%
1	1.49%	2.807	15.62	53.94%
2	1.492%	2.817	15.63	53.33%
Mean	1.493%	2.815	15.65	51.92%
Mean Rank	3	3	3	3
Sum rank	12			
change, trendscore, [160]				
0	1.519%	2.856	16.66	55.15%
1	1.489%	2.805	15.82	53.33%
2	1.611%	3.007	18.22	51.52%
Mean	1.54%	2.889	16.9	53.33%
Mean Rank	6	5	6	1
Sum rank	18			

price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, prev_price_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_change_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_trendscore_0, prev_price_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_change_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, prev_trendscore_1, [160]				
0	1.491%	2.813	15.56	42.42%
1	1.542%	2.924	16.11	50.30%
2	1.496%	2.819	15.79	50.91%
Mean	1.51%	2.852	15.82	47.88%
Mean Rank	4	4	4	7
Sum rank	19			
change, open, high, low, volume, direction, price, [160]				
0	1.623%	3.070	16.93	48.48%
1	1.493%	2.810	15.63	54.55%
2	1.498%	2.823	15.52	47.88%
Mean	1.538%	2.901	16.03	50.30%
Mean Rank	5	6	5	4
Sum rank	20			

trendscore, [160]				
0	1.754%	3.300	21.48	46.67%
1	1.488%	2.798	15.76	54.55%
2	1.557%	2.926	17.16	45.45%
Mean	1.6%	3.008	18.13	48.89%
Mean Rank	7	7	7	5
Sum rank	26			

Table 88: Full table: Predicting **next price change** on the **test** set for AAPL stock

Seed	**MAPE**	**MAE**	**MSE**	**DA**
change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	1.693%	2.657	20.98	55.15%
1	1.697%	2.664	21.05	52.73%
2	1.695%	2.66	21.01	50.91%
Mean	1.695%	2.661	21.01	52.93%
Mean Rank	1	1	1	1
Sum rank	4			
change, [160]				
0	1.713%	2.69	21.33	52.12%
1	1.709%	2.684	21.29	52.73%
2	1.713%	2.691	21.34	51.52%
Mean	1.712%	2.688	21.32	52.12%
Mean Rank	3	3	2	2
Sum rank	10			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	1.7%	2.667	21.4	50.91%

1	1.703%	2.669	21.32	53.94%
2	1.71%	2.681	21.47	50.3%
Mean	1.704%	2.673	21.4	51.72%
Mean Rank	2	2	3	4
Sum rank	11			
price, open, high, low, volume, direction, change, [160]				
0	1.711%	2.684	21.51	49.7%
1	1.722%	2.7	21.7	48.48%
2	1.709%	2.68	21.44	50.3%
Mean	1.714%	2.688	21.55	49.49%
Mean Rank	4	4	4	5
Sum rank	17			
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, prev_price_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_change_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_trendscore_0, prev_price_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_change_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, prev_trendscore_1, [160]				
0	1.728%	2.709	21.47	51.52%

1	1.756%	2.751	21.88	48.48%
2	1.694%	2.655	21.41	55.76%
Mean	1.726%	2.705	21.58	51.92%
Mean Rank	5	5	5	3
Sum rank	18			
change, trendscore, [160]				
0	1.776%	2.785	22.33	52.73%
1	1.8%	2.823	22.62	52.12%
2	1.726%	2.702	21.8	53.94%
Mean	1.767%	2.77	22.25	52.93%
Mean Rank	7	7	7	1
Sum rank	22			
trendscore, [160]				
0	1.755%	2.758	22.12	48.48%
1	1.714%	2.688	21.4	49.09%
2	1.721%	2.702	22.11	50.3%
Mean	1.73%	2.716	21.88	49.29%
Mean Rank	6	6	6	6
Sum rank	24			

Table 89: Full table: Predicting **next price change** on the **test** set for FB stock

Seed	**MAPE**	**MAE**	**MSE**	**DA**
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, [160]				
0	0.9761%	1.801	6.234	49.7%
1	0.9778%	1.804	6.253	49.09%
2	0.9751%	1.798	6.244	48.48%
Mean	0.9764%	1.801	6.244	49.09%
Mean Rank	1	1	2	4

Sum rank	8			
price, open, high, low, volume, direction, change, [160]				
0	0.9862%	1.818	6.301	52.12%
1	0.9853%	1.817	6.299	52.73%
2	0.9828%	1.812	6.292	53.33%
Mean	0.9848%	1.816	6.297	52.73%
Mean Rank	3	3	4	1
Sum rank	11			
change, trendscore, [160]				
0	0.9789%	1.806	6.176	45.45%
1	0.98%	1.808	6.178	47.27%
2	0.9791%	1.806	6.19	47.27%
Mean	0.9794%	1.807	6.182	46.67%
Mean Rank	2	2	1	7
Sum rank	12			

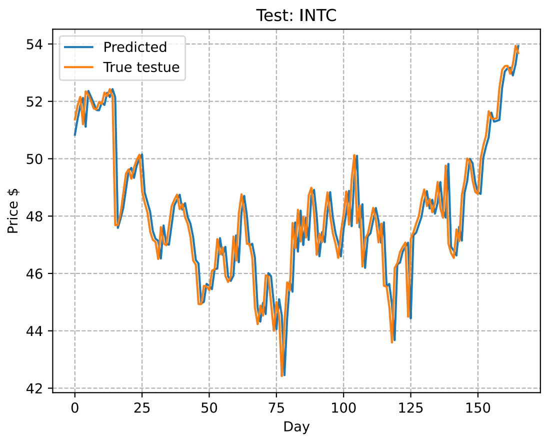
price, open, high, low, volume, direction, change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, trendscore, prev_price_0, prev_open_0, prev_high_0, prev_low_0, prev_volume_0, prev_direction_0, prev_change_0, prev_positive_0, prev_negative_0, prev_neutral_0, prev_positive_prop_0, prev_negative_prop_0, prev_neutral_prop_0, prev_trendscore_0, prev_price_1, prev_open_1, prev_high_1, prev_low_1, prev_volume_1, prev_direction_1, prev_change_1, prev_positive_1, prev_negative_1, prev_neutral_1, prev_positive_prop_1, prev_negative_prop_1, prev_neutral_prop_1, prev_trendscore_1, [160]				
0	0.9827%	1.812	6.301	48.48%
1	0.9873%	1.822	6.320	48.48%
2	0.9864%	1.819	6.313	52.73%
Mean	0.9854%	1.818	6.311	49.9%
Mean Rank	4	4	5	3
Sum rank	16			
change, [160]				
0	0.9893%	1.823	6.259	50.91%
1	0.9915%	1.828	6.274	47.88%
2	0.988%	1.822	6.254	47.88%
Mean	0.9896%	1.824	6.262	48.89%
Mean Rank	5	5	3	5
Sum rank	18			

change, positive, negative, neutral, positive_prop, negative_prop, neutral_prop, [160]				
0	0.9986%	1.844	6.331	50.91%
1	0.9778%	1.803	6.238	52.73%
2	1.022%	1.883	6.589	47.27%
Mean	0.9994%	1.843	6.386	50.3%
Mean Rank	6	6	6	2
Sum rank	20			
trendscore, [160]				
0	0.9829%	1.814	6.224	49.69%
1	0.9968%	1.838	6.374	47.88%
2	1.06%	1.961	6.752	46.67%
Mean	1.013%	1.871	6.45	48.08%
Mean Rank	7	7	7	6
Sum rank	27			

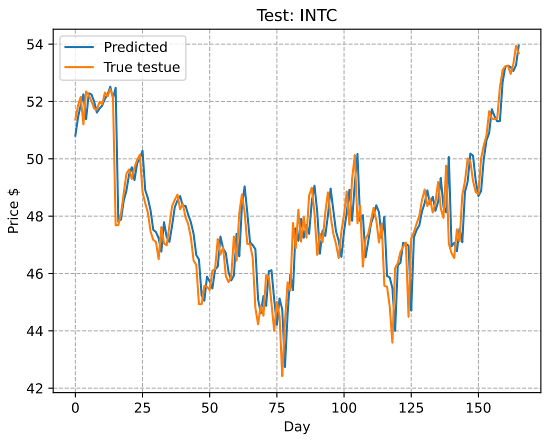
Table 90: Full table: Predicting **next price change** on the **test** set for HD stock

C Discussion

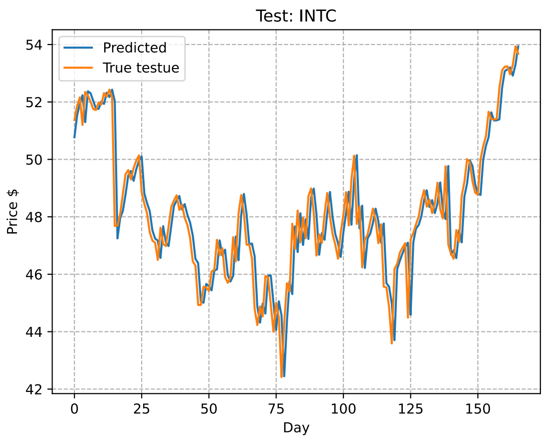
C.1 Using different data to improve prediction



(a) Using "price" only.



(b) Using all trading data.



(c) Using "price" and "trendscore".

Figure 49: Examples showing the LSTM model with context module predicting with different feature subsets on the INTC stock.

C.2 Comparing the LSTM models to the baseline models

DA

52.96%, 52.88%, 52.80%, 52.56%, 52.56%, 52.52%, 52.32%, 52.24%, 52.24%, 52.20%, 52.20%, 52.12%, 52.12%, 52.12%, 52.0%, 52.04%, 52.04%, 52.04%, 51.99%, 51.99%, 51.99%, 51.95%, 51.95%, 51.95%, 51.91%, 51.91%, 51.91%, 51.91%, 51.87%, 51.83%, 51.83%, 51.83%, 51.83%, 51.83%, 51.79%, 51.79%, 51.75%, 51.75%, 51.75%, 51.75%, 51.75%, 51.75%, 51.71%, 51.71%, 51.71%, 51.71%, 51.71%, 51.67%, 51.67%, 51.67%, 51.67%, 51.67%, 51.63%, 51.63%, 51.59%, 51.59%, 51.59%, 51.59%, 51.55%, 51.55%, 51.55%, 51.55%, 51.55%, 51.55%, 51.55%, 51.51%, 51.51%, 51.51%, 51.51%, 51.47%, 51.47%, 51.47%, 51.47%, 51.47%, 51.47%

Table 91: The top DA results related to the random model over 1000 runs. Only the results that are superior to the baseline DA are selected

MSE
139.9, 140.2, 140.7, 140.7, 140.8, 140.9, 141.0, 141.0, 141.1, 141.2, 141.2, 141.3, 141.3, 141.3, 141.3, 141.3, 141.3, 141.3, 141.4, 141.4, 141.5, 141.5, 141.6, 141.6, 141.6, 141.6, 141.6, 141.6, 141.6, 141.7, 141.7, 141.7, 141.7, 141.7, 141.7, 141.7, 141.7, 141.7, 141.8, 141.8, 141.8, 141.8, 141.8, 141.9, 141.9, 141.9, 141.9, 141.9, 141.9, 141.9, 142.0, 142.0, 142.0, 142.0, 142.0, 142.0, 142.0, 142.0, 142.0, 142.0, 142.0, 142.1, 142.1, 142.1, 142.1, 142.1, 142.1, 142.1, 142.1, 142.1, 142.1, 142.1, 142.1, 142.2, 142.2, 142.2, 142.2, 142.2, 142.2, 142.2, 142.2, 142.2, 142.3, 142.3, 142.3, 142.3, 142.3, 142.3, 142.3, 142.3, 142.3, 142.3, 142.3, 142.3, 142.3, 142.4, 142.4, 142.4, 142.4, 142.4, 142.4, 142.4, 142.4, 142.4, 142.4, 142.4, 142.4, 142.4, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.5, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.6, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.7, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.8, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 142.9, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.0, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.1, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2, 143.2

Table 92: The top MSE results related to the random model over 1000 runs. Only the results that are superior to the baseline MSE are selected

DA
62.42%, 61.21%, 60.60%, 60.0%, 60.0%, 60.0%, 60.0%, 60.0%, 59.39%, 59.39%, 59.39%, 58.18%, 58.18%, 58.18%, 58.18%, 58.18%, 58.18%, 58.18%, 57.57%, 57.57%, 57.57%, 57.57%, 57.57%, 57.57%, 57.57%, 57.57%, 57.57%, 57.57%, 57.57%, 57.57%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.96%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 56.36%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.75%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 55.15%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 54.54%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.93%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%,

53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%,
53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 53.33%, 52.72%,
52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%,
52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%,
52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%,
52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%,
52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.72%,
52.72%, 52.72%, 52.72%, 52.72%, 52.72%, 52.12%, 52.12%, 52.12%, 52.12%,
52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%,
52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%,
52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%,
52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%,
52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.12%,
52.12%, 52.12%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%,
51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%,
51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%,
51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%,
51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%,
51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%,
51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%,
51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%,
51.51%, 51.51%, 51.51%, 51.51%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%,
50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%,
50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%,
50.90%, 50.90%, 50.90%, 50.90%

50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.90%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 50.30%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.69%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 49.09%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%, 48.48%

Table 93: The top DA results related to the random model over 1000 runs on the Facebook stock. Only the results that are superior to the baseline DA are selected

MSE

20.36, 20.47, 20.48, 20.57, 20.64, 20.65, 20.67, 20.73, 20.73, 20.74, 20.74, 20.75, 20.75, 20.75, 20.76, 20.77, 20.77, 20.78, 20.78, 20.79, 20.79, 20.80, 20.81, 20.82, 20.82, 20.82, 20.83, 20.83, 20.83, 20.85, 20.86, 20.86, 20.86, 20.87, 20.88, 20.88, 20.88, 20.89, 20.91, 20.91, 20.91, 20.91, 20.91, 20.92, 20.92, 20.92, 20.92, 20.92, 20.92, 20.93, 20.93, 20.94, 20.95, 20.95, 20.95, 20.95, 20.95, 20.95, 20.96, 20.96, 20.96, 20.96, 20.96, 20.96, 20.96, 20.97, 20.97, 20.97, 20.97, 20.98, 20.98, 20.98, 20.98, 20.99, 20.99, 20.99, 20.99, 20.99, 20.99, 20.99, 20.99, 21.00, 21.00, 21.00, 21.00, 21.00, 21.00, 21.01, 21.01, 21.01, 21.02, 21.02, 21.02, 21.03, 21.03, 21.03, 21.03, 21.04, 21.04, 21.04, 21.05, 21.05, 21.05, 21.05, 21.05, 21.06, 21.06, 21.06, 21.06, 21.06, 21.07, 21.07, 21.07, 21.07, 21.08, 21.08, 21.08, 21.08, 21.09, 21.09, 21.09, 21.09, 21.09, 21.09, 21.09, 21.09, 21.10, 21.10, 21.10, 21.10, 21.10, 21.10, 21.10, 21.10, 21.11, 21.11, 21.11, 21.11, 21.11, 21.11, 21.11, 21.12, 21.12, 21.12, 21.12, 21.12, 21.12, 21.12, 21.12, 21.13, 21.13, 21.13, 21.13, 21.13, 21.13, 21.14, 21.14, 21.14, 21.14, 21.14, 21.14, 21.15, 21.15, 21.15, 21.15, 21.16, 21.16, 21.16, 21.16, 21.16, 21.16, 21.17, 21.17, 21.17, 21.17, 21.17, 21.18, 21.18, 21.18, 21.18, 21.18, 21.18, 21.18, 21.18, 21.18, 21.18, 21.19, 21.19, 21.19, 21.19, 21.19, 21.19, 21.19, 21.19, 21.19, 21.19, 21.19, 21.19, 21.20, 21.20, 21.20, 21.20, 21.20, 21.21, 21.21, 21.21, 21.21, 21.21, 21.21, 21.21, 21.21, 21.21, 21.21, 21.22, 21.22, 21.22, 21.22, 21.22, 21.22, 21.22, 21.23, 21.23, 21.23, 21.23, 21.23, 21.23, 21.23, 21.23, 21.24, 21.24, 21.24, 21.24, 21.24, 21.24, 21.24, 21.24, 21.24, 21.24, 21.25, 21.25, 21.25, 21.25, 21.25, 21.25, 21.25, 21.26, 21.26, 21.26, 21.26, 21.26, 21.26

Table 94: The top MSE results related to the random model over 1000 runs on the Facebook stock. Only the results that are superior to the baseline MSE are selected

MAE

2.598, 2.602, 2.606, 2.609, 2.614, 2.614, 2.616, 2.617, 2.621, 2.623, 2.624, 2.626, 2.627, 2.629, 2.629, 2.631, 2.632, 2.633, 2.633, 2.634, 2.634, 2.635, 2.636, 2.636, 2.637, 2.637, 2.639, 2.639, 2.640, 2.642, 2.643, 2.643, 2.643, 2.643, 2.644, 2.644, 2.645, 2.646, 2.646, 2.647, 2.647, 2.647, 2.648, 2.648, 2.648, 2.649, 2.649, 2.649, 2.649, 2.650, 2.650, 2.650, 2.650, 2.651, 2.652, 2.652, 2.652, 2.652, 2.652, 2.653, 2.653, 2.653, 2.654, 2.654, 2.655, 2.655, 2.655, 2.655, 2.655, 2.655, 2.655, 2.655, 2.656, 2.656, 2.656, 2.656, 2.657, 2.658, 2.658, 2.658, 2.658, 2.658, 2.658, 2.658, 2.659, 2.659, 2.659, 2.659, 2.659, 2.659, 2.659, 2.659, 2.660, 2.660, 2.660, 2.660, 2.660, 2.660, 2.661, 2.661, 2.661, 2.662, 2.662, 2.663, 2.663, 2.663, 2.663, 2.663, 2.663, 2.663, 2.663, 2.664, 2.664, 2.665, 2.665, 2.665, 2.665, 2.665, 2.665, 2.666, 2.666, 2.666, 2.666, 2.666, 2.666, 2.666, 2.666, 2.667, 2.667, 2.667, 2.667, 2.667, 2.667, 2.667, 2.667, 2.667, 2.668, 2.668, 2.668, 2.668, 2.668, 2.668, 2.668, 2.668, 2.669, 2.669, 2.669, 2.669, 2.669, 2.669, 2.669, 2.669, 2.669, 2.669, 2.669, 2.669, 2.670, 2.670, 2.670, 2.670, 2.670, 2.670, 2.670, 2.670, 2.670, 2.670, 2.671, 2.671, 2.671, 2.671, 2.671, 2.671, 2.671, 2.671, 2.671, 2.671, 2.672, 2.672, 2.672, 2.672, 2.672, 2.673, 2.673, 2.673, 2.673, 2.673, 2.673, 2.674, 2.674, 2.674, 2.674, 2.674, 2.674, 2.675, 2.675, 2.675, 2.675, 2.675, 2.675, 2.675, 2.675, 2.675, 2.675

Table 95: The top MAE results related to the random model over 1000 runs on the Facebook stock. Only the results that are superior to the baseline MAE are selected

MAPE

1.655%, 1.657%, 1.663%, 1.665%, 1.666%, 1.666%, 1.668%, 1.671%, 1.672%, 1.673%, 1.675%, 1.676%, 1.676%, 1.676%, 1.676%, 1.677%, 1.677%, 1.677%, 1.678%, 1.678%, 1.678%, 1.679%, 1.680%, 1.680%, 1.680%, 1.681%, 1.682%, 1.682%, 1.682%, 1.683%, 1.683%, 1.684%, 1.684%, 1.684%, 1.684%, 1.685%, 1.686%, 1.686%, 1.686%, 1.686%, 1.687%, 1.687%, 1.687%, 1.688%, 1.688%, 1.688%, 1.688%, 1.688%, 1.689%, 1.689%, 1.689%, 1.689%, 1.690%, 1.690%, 1.690%, 1.690%, 1.690%, 1.690%, 1.691%, 1.691%, 1.691%, 1.692%, 1.692%, 1.692%, 1.692%, 1.692%, 1.692%, 1.692%, 1.692%, 1.692%, 1.692%, 1.693%, 1.693%, 1.693%, 1.693%, 1.693%, 1.693%, 1.693%, 1.693%, 1.693%, 1.694%, 1.694%, 1.694%, 1.694%, 1.694%, 1.694%, 1.695%, 1.695%, 1.695%, 1.695%, 1.695%, 1.695%, 1.695%, 1.695%, 1.696%, 1.696%, 1.696%, 1.696%, 1.696%, 1.696%, 1.696%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.697%, 1.698%, 1.698%, 1.698%, 1.698%, 1.698%, 1.698%, 1.698%, 1.699%, 1.699%, 1.699%, 1.699%, 1.699%, 1.699%, 1.699%, 1.699%, 1.699%, 1.699%, 1.699%, 1.700%, 1.700%, 1.700%, 1.700%, 1.700%, 1.700%, 1.700%, 1.700%, 1.701%, 1.701%, 1.701%, 1.701%, 1.701%, 1.701%, 1.701%, 1.701%, 1.701%, 1.701%, 1.701%, 1.702%, 1.702%, 1.702%, 1.702%, 1.702%, 1.702%, 1.702%, 1.702%, 1.702%, 1.702%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.703%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.704%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%, 1.705%

Table 96: The top MAPE results related to the random model over 1000 runs on the Facebook stock. Only the results that are superior to the baseline MAPE are selected

DA

53.37%, 53.01%, 52.80%, 52.76%, 52.68%, 52.64%, 52.60%, 52.48%, 52.48%,
52.40%, 52.36%, 52.36%, 52.36%, 52.32%, 52.28%, 52.28%, 52.28%, 52.16%,
52.12%, 52.12%, 52.12%, 52.12%, 52.12%, 52.08%, 52.04%, 51.99%, 51.95%,
51.95%, 51.95%, 51.95%, 51.95%, 51.95%, 51.91%, 51.91%, 51.91%, 51.91%,
51.87%, 51.87%, 51.87%, 51.87%, 51.83%, 51.83%, 51.83%, 51.79%, 51.79%,
51.79%, 51.75%, 51.75%, 51.75%, 51.75%, 51.75%, 51.71%, 51.71%, 51.71%,
51.67%, 51.67%, 51.67%, 51.67%, 51.67%, 51.63%, 51.63%, 51.63%, 51.63%,
51.63%, 51.59%, 51.59%, 51.59%, 51.59%, 51.59%, 51.55%, 51.55%, 51.55%,
51.55%, 51.55%, 51.55%, 51.55%, 51.51%, 51.51%, 51.51%, 51.51%, 51.51%,
51.47%, 51.47%, 51.47%, 51.47%, 51.47%, 51.47%, 51.47%

Table 97: The top DA results related to the random model with a 51.47% chance of predicting a higher price than the current over 1000 runs on the test set. Only the results that are superior to the baseline DA are selected

Phi Thien & Jonas Laskemoen

NTNU
Norwegian University of
Science and Technology