STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **9**, June 2021, pp 268–287.
Published online in International Academic Press (www.IAPress.org)

Stock Price Predictions with LSTM Neural Networks and Twitter Sentiment

Marah-Lisanne Thormann[1], Jan Farchmin[1], Christoph Weisser[1,3], René-Marcel Kruse[2], Benjamin Säfken[2,3], Alexander Silbersdorff[1,3]

[1]*Faculty of Economics, University of Göttingen, Germany*
[2]*Centre for Statistics, University of Göttingen, Germany*
[3]*Campus Institute Data Science, Göttingen, Germany*

Abstract Predicting the trend of stock prices is a central topic in financial engineering. Given the complexity and non-linearity of the underlying processes we consider the use of neural networks in general and sentiment analysis in particular for the analysis of financial time series. As one of the biggest social media platforms with a user base across the world, Twitter offers a huge potential for such sentiment analysis. In fact, stocks themselves are a popular topic in Twitter discussions. Due to the real-time nature of the collective information quasi contemporaneous information can be harvested for the prediction of financial trends. In this study, we give an introduction in financial feature engineering as well as in the architecture of a Long Short-Term Memory (LSTM) to tackle the highly nonlinear problem of forecasting stock prices. This paper presents a guide for collecting past tweets, processing for sentiment analysis and combining them with technical financial indicators to forecast the stock prices of Apple `30m` and `60m` ahead. A LSTM with lagged close price is used as a baseline model. We are able to show that a combination of financial and Twitter features can outperform the baseline in all settings. The code to fully replicate our forecasting approach is available in the Appendix.

Keywords Twitter, TextBlob, Stock Price, Prediction, Neural Networks, LSTM, RNN

DOI:10.19139/soic-2310-5070-1202

1. Introduction

“The most valuable commodity [...] is information.” This famous quote by Gordon Gekko in The Wall Street in 1987 implies that in the framework of the stock market profits can be generated based on superior knowledge. This stands in contradiction to the Efficient Market Hypothesis (EMH) which “states that market price mirrors the assimilation of all the information available. As the new information enters the market the system immediately enters the unbalanced state and predicted correct change is eliminated by the new price. Hence given the information it is not possible to predict the future price of the stock.”[2] However, during the last years the EMH was frequently taken into question as some researchers found evidence against it. Some recently published approaches show that the prediction of stock prices can be significantly improved by the use of knowledge from micro blogs like Twitter. [8] achieves an accuracy of 87.6 % in predicting the daily movements of the Dow Jones Industrial Average (DJIA) by using a large-scale of tweets. Among others [36] shows that this accuracy level is difficult to duplicate in different settings.

A crucial point in our framework is the proper feature engineering with tweets. There are many different possibilities to extract Twitter data and to create variables based on tweets. For example, one could use the data to assign tweets into different emotional categories or to use it only for counting certain words of interest. Depending on the chosen method, the predictive power of the model might differ strongly. [8, 36, 11, 55, 38, 59, 44, 42, 43]

*Correspondence to: Christoph Weisser (Email: c.weisser@stud.uni-goettingen.de). Campus Institute Data Science, Göttingen, Goldschmidtstraße 1, 37077 Göttingen, Lower Saxony, Germany

ISSN 2310-5070 (online) ISSN 2311-004X (print)

Furthermore, the applied statistical method has a high impact on the achievement as the prediction problem is more complicated than a simple linear regression issue due to the non-stationary stock price data. Therefore, [42], [47], [11], [40], [14] use neural networks (NN) which are powerful tools to tackle highly nonlinear problems. More specifically, recurrent neural networks (RNN) are applied as those networks are known to be a very good choice for time series data. In order to address the issue of exploding or vanishing gradients Long-Short-Term-Memory (LSTM) framework is used.[17]

The structure of this paper is organized as follows, firstly we provide a literature review regarding NN and Twitter data for stock price (index) predictions. Afterwards, in a section about feature engineering, we discuss various variables which will be relevant for the later empirical analysis. The focus will be especially on the Twitter covariates which will be calculated partly with the help of the Python package `TextBlob`. Thirdly, we provide insights into the RNN and the LSTM approach respectively. Moreover, we introduce Random Search as a sophisticated solution for optimizing hyperparameters. Finally, in our empirical analysis we forecast stock prices `30m` and `60m` ahead for Apple. This research contributes to the literature with a practical guide to forecast short-term stock prices with a LSTM including `TextBlob`-Twitter variables. The code to fully replicate our forecasting approach is available in the Appendix.

2. Literature Review: Stock Price Predictions

2.1. *Neural Networks*

As stated in the introduction, NN are powerful tools to model highly nonlinear problems and to capture complex relationships between variables which allow for the flexible modelling of stock price time series. Various forms of NN exist, but during the past years two specific types have been established are in particular frequently applied. Firstly, the long short term memory (LSTM) which belongs to the class of recurrent neural networks (RNN) and secondly the convolutional neural network (CNN).

Starting with the first mentioned network, [42] analyze various machine learning algorithms for daily stock market predictions and concludes that an approach based on LSTMs works best. Also [47] uses a LSTM for predicting the National Stock Exchange Fifty (NIFTY 50). By trying different combinations of open, close, high and low prices as input parameters, they confirm the usefulness of the LSTM for stock price predictions. Further approval can be found in [11] where daily stock prices of Apple, Microsoft and Google are successfully predicted with an LSTM. The results of [40] for predicting the share prices of Apple, Google and Tesla show once again that a LSTM seems to be more accurate than the traditional machine learning algorithms. Moreover, [14] compares the LSTM against the standard deep net and logistic regression for S&P 500 predictions from December 1992 until October 2015 and finds that the LSTM beats both approaches by a clear margin. Furthermore, [37] concludes that a LSTM performs better in prediction of the close price of iShares MSCI United Kingdom ETF than other machine learning algorithms. Finally, [53] compares several regression techniques for predicting stock prices in short intervals and concludes that LSTM outperforms the other models by a large margin.

In recent years the usage of the class of CNN for modelling financial time series received more attention in the literature. [13] compares different NN for daily S&P 500 predictions and derives that CNN architectures perform best for financial time series. [52] shows that CNNs have the best predictive qualities for stock prices. Additionally, [51] confirms with their study the high accuracy of CNNs in the prediction framework of three Thai stocks. Moreover, [22] uses various NNs to predict stock prices of five different companies from National Stock Exchange (NSE) and New York Stock Exchange (NYSE) and finds that the CNN is outperforming the other models.

Most recently, the literature has shown a drive towards the combination of LSTM and CNN for stock price predictions. Two recent papers are [25] and [28] where both conclude that the hybrid form outperforms the competitors.

2.2. *Twitter*

In the following, we focus on publications that use Twitter data for stock price predictions. Considering the data collection, two main approaches can be found in the literature that are both based on the Twitter API. On the one hand, in line with [11], [55] or [38] one can limit the extraction by defining certain search words, like for example the company name and ticker symbol to stream only tweets which are directly connected to the stock in question. [55] even analyzes the performance between different keywords and concludes that the predictive power for streamed tweets with the company ticker symbol provides better performance for Apple stock prices. However, using this approach the total quantity of tweets per day is lower compared to streaming tweets with the full company name.

On the other hand, a further frequently employed method is the usage of geographical coordinates, e.g. just tweets from major cities in the United States, or emotional keywords, e.g. "I feel" or "I am", and to stream all tweets that are available.[†] [8], [59] and [36] analyzes Twitter text content and gathers them into mood categories. Their results show a correlation between the Dow Jones Industrial Average and these categories. Further, their findings suggest that the accuracy of stock price indices prediction be significantly improved by using the mood categories.

In the following, we discuss the data processing in order to generate valuable features. The majority of publications directly focus on the evaluation of the sentimental content. However, only a few papers – like [33] – use meta data beyond the expressed opinion of the user, like count of tweets per day. In the framework of the sentimental evaluation, there are two major ideas. Firstly, one frequently used approach is the classification of the tweets into multiple mood categories. In this context, probably the most well known example for this method is the publication of [8]. By classifying tweets into the six mood categories calm, alert, sure, vital, kind, and happy they receive especially for certain moods significant results. The second popular method is a more simple binary classification of the tweets into positive and negative. [8] try this technique as well, but in contrast to the mood analysis do not find any predictive power for daily DJIA prices.

[11] uses an extended version of the positive/negative classification by introducing seven different categories from very negative to very positive and confirmed predictive power for daily stock prices of Apple, Microsoft and Google. Further confirmation for this finding can be found in [38], where tweets are only classified into three categories (positive, neutral and negative) and predictive power for daily up/or down predictions for the Microsoft stock is concluded. [59] provides similar results with predictions for different tech companies based on positive and negative sentiment calculations. Moreover, a hybrid between both methods is used in [44] where a classification into positive and negative tweets is performed by counting emoticons or certain emotional keywords. [10] successfully predicts daily stock prices by using only two moods, e.g. happy and sad. The happiness mood is also a promising variable in the analysis by [36] who calculates in total four different mood categories and is thereby able to improve daily predictions of the DJIA.

Only few publications refute the predictive power of twitter variables for stock price predictions. For example, [43] classifies tweets into 8 mood categories and assesses the improvement of the daily predictions of S&P 500 and DJIA as negligible. Additionally, [42] finds that only extreme polarities in tweets have a significant influence.

3. Feature Engineering

For our empirical analysis, we distinguish between "classical" financial variables that are accessible for instance via Yahoo Finance and "novel" covariates derived from Twitter.

3.1. Yahoo Finance Features

Starting with first mentioned feature category, we use five common variables provided by Yahoo Finance:

OHLC. Yahoo Finance offers Open, High, Low and Close price (OHLC) for each stock per time interval. Each type of price serves as a single input.

[†]This could be done for example by the Python Package TTLocVis [26].

Volume. Additionally, we use the trading volume in a `30m` interval. It reflects the current trading activities of a stock and therefore may help to improve predictions. The trading volume is used for example in the analysis by [1].

3.2. *Twitter Features*

Concerning the features based on tweets, we distinguish between two different kinds of covariate types. On the one hand, we want to capture the content of the Twitter data by calculating sentiments. On the other hand, we construct predictors which ignore the sentimental content of the tweets and only concentrate on descriptive statistics, e.g. number of tweets per time interval.

As shown in the literature review, many publications analyze the predictive power of sentiment based variables. However, we contribute with a more fundamental analysis by extracting more information from the available data. We assess whether more simple Twitter features could already improve the prediction of a NN, which is shown by [12], but only for a simple Support Vector Machine model.

3.2.1. Content Unrelated Variables Firstly, we assess which content unrelated information can be extracted from the data. We use descriptive statistics to capture different attributes of the data. Note, that we introduce our variables regarding stock data in the interval of `30m` as this was the interval of our analyzed data. Let $y_{i,d,t}$ be the close price of company i on day d at time t with $t \in \{10.00 \text{ am}, \ \dots \ , \ 4.00 \text{ pm}\}$. Then $x_{i,k,d,t}$ is the k^{th} Twitter variable of company i on day d with $k = 1, ..., 12$ calculated based on $TW_{i,d,[t-31,\ t-1)}$ which is the subset of tweets for company i on day d from time $t - 31$ minutes till time $t - 1$ minute. Furthermore, let $tw_{i,d,n}$ be the n^{th} tweet on day d for company i with $n = 1, ..., N$, where N is the total number of tweets for the company on that day.

$$x_{i,1,d,t} = |TW_{i,d,[t-31,\ t-1)}| \tag{1}$$

$$x_{i,2,d,t} = \frac{x_{i,1,d,t}}{30} \tag{2}$$

Frequency. One simple but plausible variable is the count of released tweets per `30m`, which is shown in equation (1). In addition, we calculate the average number of released tweets per `1m` during the `30m` time horizon, illustrated in equation (2). It seems to be reasonable that the underlying information can offer predictive power as one would expect that the number of tweets increases, when there are positive or especially negative events for a company. [33] already observes relationship.

Volatility. As shown in equation (3), we include the volatility of the released tweets as a predictor. We calculate the number of tweets per `1m` and compute the volatility during the overall `30m` time period. For the tweets per `1m` we maintain sufficient number of values for the volatility calculation. For instance, if one uses the number of tweets per `15m`, there would be only two values available for calculating the variance during the `30m` time horizon of the stock data. In general, the idea behind this feature is similar to the frequency variable. We assume that on average the volatility should be comparable on a daily or weekly basis if nothing extraordinary happens to the company.

Max (Min) number of tweets. Additionally, we include the minimum and maximum number of tweets per `1m` during the `30m` stock data interval which is shown in equation (4). The features are used to provide additional information for the model, regarding the distribution of tweets during the `30m` horizon. For example, if there is a sudden peak for the number of tweets per `1m`, the average of tweets per `1m` might not be able to capture this information completely, when all other values are very similar.

$$x_{i,3,d,t} = \frac{1}{30-1}\sum_{m=0}^{29}(c_{i,d,t,m} - x_{i,2,d,t})^2 \quad \text{with } c_{i,d,t,m} = \sum_{n=1}^{N} \mathbb{1}(tw_{i,d,n} \in TW_{i,d,[t-31+m,\ t-31+m+1)}) \tag{3}$$

$$x_{i,4,d,t} = \max c_{i,d,t,m} \quad \text{and} \quad x_{i,5,d,t} = \min c_{i,d,t,m} \tag{4}$$

Length of tweets. The last feature is the average length of the tweets during the time interval. It is mainly motivated by the results of [12], who conclude that this variable offers less predictive power than for example the count of tweets. The general argument is, that tweets tend to get longer when there are some news regarding a company as the users have new topics to discuss.

3.2.2. Content Related Variables After determining content unrelated variables, we focus on features that measure the content of a tweet. As shown in the literature review, two frequently used approaches are the assignment of multiple mood categories or a classification into positive or negative tweets. The first mentioned categorization is mainly used for stock indices and the second for stock prices. Therefore, as the close prices of Apple were our prediction targets, we decided to use a sentiment classification.

Some publications, for example [59] and [38], manually label the Twitter data. However, pretrained classifiers such as those provided by the Python package `TextBlob` ([30]) already offer a very effective implementation for sentiment predictions. The calculation of following variables are based on the analysis of [21].

Polarity. First, we calculate the average polarity based on the sentiment attribute of `TextBlob` for all tweets that are allocated to a `30m` time interval. The polarity score ranges from -1 to 1. Scores greater than zero indicate positive content and values below zero negative content. If the score is equal to zero, the tweet is classified as neutral. In order to compute the polarity, we employ `TextBlobs` inbuilt dictionary approach, where each sentimental word, like "good" or "bad", has its own score as illustrated in 1. If there are several sentimental words in one text, `TextBlob` calculates in general the simple average. However, deviating calculations can occur in the case of amplifications, e.g very good, and negations, e.g. not good. This variable is motivated by the assumption, that when the average sentiment is positive, the stock price should rise and vice versa.

Subjectivity. Second, in a similar fashion we compute the average subjectivity. The subjectivity score ranges from 0 to 1. Scores close to zero indicate objective content and values close to one subjective content. The calculations are again based on a dictionary approach. A similar example as seen for the polarity can be found in 2. The main idea behind this variable, is to have a counterpart for the polarity variable, as some tweets might be very positive, however, the used language may indicate that the tweet could be rather subjective.

$$x_{i,10,d,t} = |TW_{i,d,[t-31,\ t-1),pos}| \quad \text{and} \quad x_{i,11,d,t} = |TW_{i,d,[t-31,\ t-1),neg}| \tag{5}$$

Number of positive (negative) tweets. Third, we classify the single tweets into positive, negative or neutral groups based on their polarity scores. Afterwards, we count the number of positive and negative tweets in a `30m` time interval. The cardinality of this subset is given by, let $TW_{i,d,[t-31,\ t-1),pos}$ be a subset of $TW_{i,d,[t-31,\ t-1)}$ with all positive classified tweets, then $x_{i,10,d,t}$ as shown in equation (5). Analogously, $x_{i,11,d,t}$ is the cardinality of the subset of all negative classified tweets. The notion is to have additional information on the amount of positive or negative classifications, since the previous described average polarity only ranges between -1 and 1.

Share of positive (negative) tweets. Finally, the previously described variables are extended by introducing percentage shares for positive and negative tweets, in comparison to the total number of tweets per time interval.

4. Methodology

4.1. Model Framework

4.1.1. RNN The usage of a classical NN is limited because it takes a fixed-sized vector as input and produces a fixed-sized vector as output. A RNN is an extension that is able to process sequential data. The RNN iterates through a sequence and produces output at each time step. During this process, the network keeps an additional state besides the output to save information about historic values. These memory units enable the RNN to model short term dependencies. The model can take the ordering of the elements and the relations between the elements of a sequence into account. A basic illustration of the approach can be found in Figure 1. A basic RNN consists of six major components. At each time point t the model receives a new network input which is denoted by $\boldsymbol{x}_t$. Then the internal network state at time t, i.e. $\boldsymbol{s}_t$, is generated based on the weighted network input ($\boldsymbol{x}_t\boldsymbol{U}$) and the weighted previous internal state ($\boldsymbol{s}_{t-1}\boldsymbol{W}$). By combining them in an activation function $\phi_s(\boldsymbol{x}_t\boldsymbol{U} + \boldsymbol{s}_{t-1}\boldsymbol{W})$ one receives $\boldsymbol{s}_t$. $\boldsymbol{U}$ defines the contribution of $\boldsymbol{x}_t$ for $\boldsymbol{s}_t$ and $\boldsymbol{W}$ the contribution of $\boldsymbol{s}_{t-1}$ for $\boldsymbol{s}_t$. Further, y_t denotes the network output at time t and is generated by the weighted internal state at this time transformed by a second activation function $\phi_y(\boldsymbol{s}_t \boldsymbol{V})$. Therefore, the contribution of $\boldsymbol{s}_t$ for y_t is defined by $\boldsymbol{V}$. Even though this approach is more sophisticated for processing time series data than classical NNs, it still has a major drawback known under the vanishing and exploding gradients problem. During the processing of longer input sequences an activation function could produce a very small gradient, which indicates low importance. Therefore, the RNN forgets about this step. [58]

4.1.2. LSTM LSTM models are very popular NNs for handling sequential data, e.g. stock prices. The layer of a LSTM consists of a set of recurrently connected blocks, the so called cell state or memory blocks. Every single block contains multiplicative units the forget, input, output and update gate as well the weight parameter for the input, output and internal state. Because of this architecture, the LSTM is able to do continuous analogue read, write and reset operations at the current LSTM cell. For this reason a LSTM overcomes the problem of vanishing and exploding gradients, and is able to model temporal sequences and long-range dependencies more accurately than conventional RNN's. [23, 19, 50, 32]

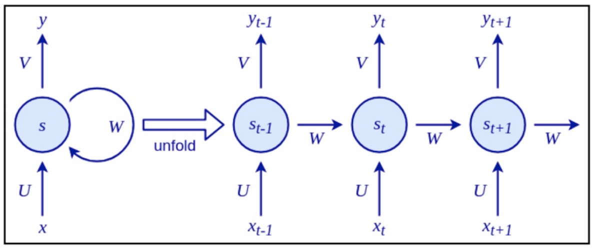

Figure 1. Illustration of a RNN where the right side shows the unfolded version and the left side the folded presentation of the model.
Source: [58], p. 198.

Figure 2 shows the basic components of a LSTM cell. The actual cell state is noted by c_t. This is the LSTM's memory, which stores the data along with all the cells. Based on the interaction with the gate structure it modifies the information in the cell state of each cell. Every LSTM cell receives as input the previous hidden state h_{t-1}, the previous cell state c_{t-1} and the current input x_t. The first sigmoid layer is also called the forget gate (6) because the output selects the amount of information to be included from the previous cell in the current cell.

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \tag{6}$$

The output is a number in the interval [0,1] which is pointwise multiplied with the previous cell state. If the gate generates a zero, no new data will be added to the cell state, and by generating a one the full data will be added. Through this process, the gate decides which information should be kept or forgotten, which gives it the name forget gate.

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{7}$$

Like the first sigmoid layer, the second sigmoid layer (7) takes as input the hidden state of the previous cell and combines it with the current input.

$$c'_t = tanh(w_c[h_{t-1}, x_t] + b_c) \tag{8}$$

$$c_t = f_t * c_{t-1} + i_t * c'. \tag{9}$$

To regulate the network, the previous hidden state, and current input also go into the tanh layer (8). Afterwards, the output of the tanh and sigmoid function gets multiplicated pointwise, where the sigmoid output decides which information is important to keep from the tanh output. For this reason, the layer is also called input gate (9) .

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{10}$$

$$h_t = o_t * tanh(c'_t) \tag{11}$$

The output of the current cell state (11) is also the result of a pointwise multiplication of a tanh and a sigmoid layer. Which part of the cell state will be presented in the output will be determined by the sigmoid layer, whereas the tanh layer shifts the output in the range of [-1,1]. These gates determine which data goes into the next step, and regulate the optimization of the weights. If the gradient vanishes during backpropagation, it will also vanish during forward propagation. Therefore, this weight will also not impact the estimation, so it will not be optimized within backpropagation. [58]

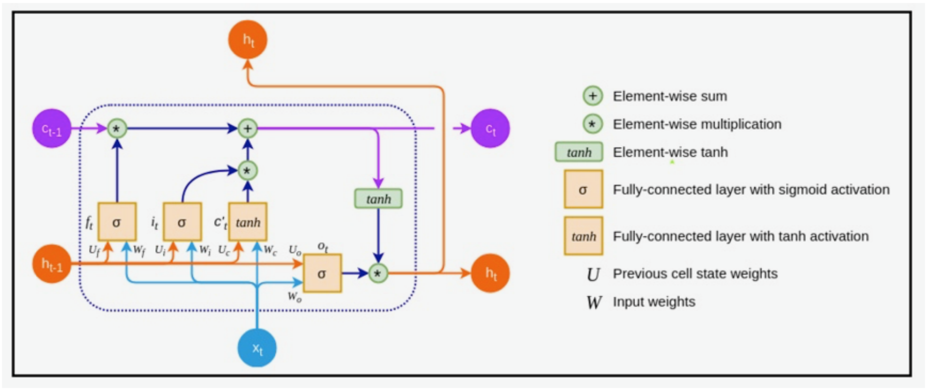
Element-wise sum
Element-wise multiplication
tanh Element-wise tanh
σ Fully-connected layer with sigmoid activation
tanh Fully-connected layer with tanh activation
U Previous cell state weights
W Input weights
h_t
c_{t-1}
c_t
h_{t-1}
x_t
f_t
i_t
c'_t
o_t
U_f W_f U_i W_i U_c W_c U_o W_o

Figure 2. Illustration of LSTM Cell. *Source:* [58], p. 210.

4.2. Hyperparameters

As optimization and selection of the hyperparameters play an important role in the performance of a LSTM, the following part gives an overview of how we chose the tuning parameteres. Hyperparameters are manually set training variables that are determined before training the model. One major problem of NNs are the over- and underfitting. On the one hand, if the model overfits, the NN performs very good on the training and validation dataset, but performs very poorly on out of sample data. Due to a high number of parameters and, therefore, complexity the model has then become so effective in explaining the specific characteristics of the training set that it has lost the ability to generalize. As a result, the model will not be able to make stable and consistent predictions for new data. On the other hand, if a model has a more simple structure, it tends to underfit the data because it overgeneralizes the given input. The previous issue refers to the so-called bias-variance trade off where a high variance is usually proportional to overfitting and a higher bias to underfitting [45, 49].

For finding a good balance between bias and variance in the framework of a LSTM, it is important to select the main hyperparameters carefully which are the number of epochs, batch size, activation function, optimizer, and number of hidden layers and neurons. The epoch size determines the number of times that the learning algorithm will work through the entire training dataset. Thus, the number of epochs can be any integer between one and infinity where lower values tend to create underfitting and bigger epoch sizes lead to an overfitting problem as the learning algorithm will have too many rounds to over-optimize the weights of the LSTM. A training step can be further split into many iterations based on the batch size. It defines the number of samples to work through before the internal parameters of the model are updated. Therefore, the batch size has a huge impact on how quickly a model learns and on the stability of the learning process. Often smaller batch sizes are used because they are noisy, providing a regularizing effect and a decreased error in generalization. [62] Due to this fact we decide to use 32 batches which is also confirmed as a suitable default by [34] and Bengio13practicalrecommendations.

To increase the performance of the model the right optimizer plays an important role as well. There are many variants that could be used in our framework. A common optimizer class is the stochastic gradient descent (SGD). Variants of the vanilla SGD are momentum optimization, Nesterov accelerated gradient, or adaptive learning-rate methods, i.e. AdaGrad, Adam or RMSProp. We will not go into detail regarding the differences, however a detailed description of the different approaches can be found in [17]. In our empirical analysis we use Adam (adaptive moment estimation), which works well in practice and outperforms other adaptive learning-method algorithms. Therefore, the optimizer is used among others in [13], [37] and [27]. A reason for this good performance is that Adam combines the benefits of both the AdaGrad and RMSProp algorithms. [29, 48]

$$
\begin{aligned}
\boldsymbol{m} &= \beta_1 \boldsymbol{m} - (1-\beta_1)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\
\boldsymbol{s} &= \beta_2 \boldsymbol{s} + (1-\beta_2)(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}))^2 \\
\hat{\boldsymbol{m}} &= \frac{\boldsymbol{m}}{1-\beta_1^t} \\
\hat{\boldsymbol{s}} &= \frac{\boldsymbol{s}}{1-\beta_2^t} \\
\boldsymbol{\theta} &= \boldsymbol{\theta} + \eta \frac{\hat{\boldsymbol{m}}}{\sqrt{\hat{\boldsymbol{s}}+\epsilon}}
\end{aligned}
\tag{12}
$$

Equation (12) shows the main steps of Adam. One can see that an exponential moving average of the gradient of the cost function with respect to the weights $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ captured by $\boldsymbol{m}$ and the squared gradient $(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}))^2$ captured by $\boldsymbol{s}$ is calculated by the Adam optimizer. These moving averages are estimates of the first and second moment of the gradient and are initialized as vectors of $0's$. This leads to moment estimates which are biased during the initial timesteps, and when the decay rates are small. To counteract this bias the bias-corrected estimates $\hat{\boldsymbol{m}}$ and $\hat{\boldsymbol{s}}$ are introduced. The parameters β_1 and β_2 control the decay rates of these moving averages and t represents in this formula the iteration number starting at one. The vector of weights $\boldsymbol{\theta}$ is then updated by the division of $\boldsymbol{m}$ by the square root of $\boldsymbol{s}$ and a smoothing term ϵ, which is a very small number to prevent any division by zero. The quotient is afterwards weighted by the learning rate η [29, 17, 48]. Proceeding, a loss function has to be determined which will be the minimization target of the optimizer. We decide to use the MSE, which is shown in equation (13), as it is a popular choice for the evaluation of financial time series predictions. In this formula N stands for the total number of observations. The MSE is frequently used in other publications such as in [39], [57], [31], [56], [13], [37], [55], [4] and [54].

$$
\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2 \tag{13}
$$

Moreover, the number of hidden layers and neurons can have a huge impact on the success of the predictions. A higher number for at least one of the parameters makes it possible to cover a larger complexity of the relationship between input and output variables. However, one has to pay attention the model does not over- or underfit. Therefore, finding the right combination can be a challenging task. Among other [17] recommends to construct complex models with many parameters and then to apply regularization techniques. Regularization incorporates different approaches to enable neural networks to choose models that generalize well. The aim is to minimize the variance for more accurate predictions on new input sets, without increasing the bias due to a systematic failure.

On the one hand, we decide to implement early stopping which is a method that makes it possible to specify an arbitrarily large number of training epochs. It will stop the training if no further significant improvements on the validation set are achieved and automatically remembers the round that led to the best result.[7, 17, 18, 54] On the other hand, a second commonly used regularization method that we implement are dropout layers which also help to reduce the problem of overfitting. While training a network, the dropout method excludes the data on the input connection for a given probability so that the data cannot be used for the next step. In Keras when creating a LSTM layer, this is specified with a dropout argument. The dropout value is a percentage between 0 – no dropout – and 1. [61, 54, 16, 41]

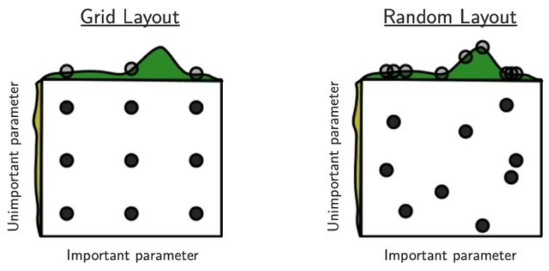

Figure 3. Gird Search vs Random Search.
Source: [6], p. 284.

Nevertheless, even though simpler models could potentially be ignored through regularization, there remains an infinite number of more complex models that can be constructed. Due to this fact, the random search offers a handy implementation of trying various tuning parameter combinations for receiving the most promising architecture. This method is known to be more efficient than a grid search as not all hyperparameters are equally important which is illustrated in Figure 3.

Additionally, trying all constellations can be very time consuming, especially for a high number of parameters with large ranges. In general, the user defines a flexible architecture with a predefined range for all tuning parameters and a certain number of trials. Afterwards, for each trial random search decides independently on the parameter combination and fits the model various times. It makes sense to fit the same model several times as the weights at the beginning are randomly initialized. [6, 20, 15]

We use a random search with 50 maximal trials and 3 executions per trial. The first LSTM cell can have recurrent units from 17 to 500 and as an activation function the common options relu, tanh and sigmoid. For the number of hidden LSTM layers we define a range between 0–3 where each layer has the possibility of having between 17 and 250 recurrent units. If the random search specifies a model with hidden layers, there is also a hidden dropout per layer with a minimum dropout of 5% and maximum of 95%.

5. Empirical Analysis

5.1. Data preparation

Two different data sources are used in this study. On the one hand, we stream the stock prices for Apple via the `yfinance` API in Python. On the other hand, we create Twitter Developer Accounts to stream past Twitter data via the `tweepy` ([46]) API in Python. As shown in the literature review for stock price predictions based on Twitter variables, a frequently used approach to extract tweets for a company is to use the respective ticker symbol (e.g. `AAPL`) and cashtag (e.g. `$AAPL`). We use both as search words in separate queries.

As part of the data cleaning, we remove mentions, hashtags, retweets and URLs from the collected Twitter data. We decide against further preprocessing, since we use the Python package `TextBlob` ([30]), where further prepossessing is not necessarily useful. For example, by using stemming `TextBlob` cannot recognize some words anymore as it is using a dictionary approach based on full words. Lemmatization can be problematic as well because it groups inflected forms of a word to the same expression. For example, "better" would have been changed to "good". However, `TextBlob` has different polarity and subjectivity scores for both words, so that we decide against lemmatization. We use the polarity and subjectivity attribute of `TextBlob` to calculate scores per cleaned tweet. Afterwards, we classify the tweets into positive (`score > 0`), negative (`score < 0`) and neutral (`score == 0`) based on the polarity.

Since we analyze intervals of `30m`, all further explanations will refer to this period length. The tweets are assigned to the stock prices via a moving window approach. After every `30m` we receive a new close price. The tweets of this time interval are used for the calculation of the Twitter features. We have to point out that the previous described procedure remains the same for the beginning of the week and of a trading day. As such, we lose tweets on the weekend or during the night.

5.2. Results

We use a Random Search with the same range of hyperparameters. We separate the collected data into training, validation and test sets. For the two first mentioned categories we use data from 30th November 2020 until 15th January 2021. Furthermore, we apply the very common training-validation-split rule of 80/20. For the test set we take the remaining data from 16th January 2021 to 31st January 2021. An illustration of the number of tweets per day for the ticker symbol is shown in Figure 4 and for the cashtag in Figure 5. It can be seen, that the daily tweet number for `AAPL` is higher than for `$AAPL`. However, both search words show a similar pattern over time. On the weekends the daily number decreases in comparison to weekdays. In addition to that, both figures display a peak a few days before Christmas and an increased volume for the last days in January. The latter could be explained by the discussions over WallStreetBets and GameStop on Twitter where many tweets included various cashtags or ticker symbols.

For the ticker symbol one further peak can be observed at the beginning of 2021 which could be related to the news that Apple will release an autonomous electric car in the next five to seven years.[‡]

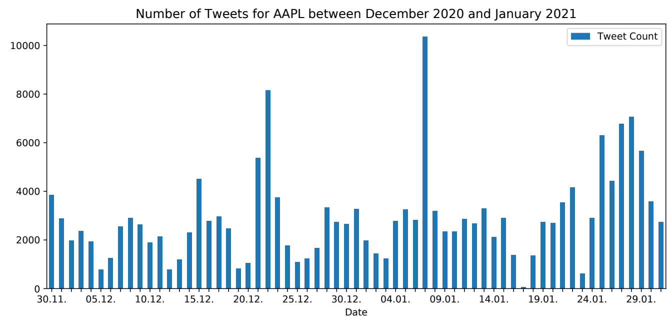

Figure 4. Number of Tweets per day for AAPL.
Source: Own Representation.

We start with our baseline model which is a LSTM with lagged close price as the only variable. Proceeding, we combine this baseline feature with every other lagged covariate separately. Afterwards, we take the variable combination which leads to the best result and combine this selection with all other remaining covariates again separately. Subsequently, we take again the best combination and proceeded in a similar fashion as before until no further improvement in the loss function is observable. We decided for this procedure because we have 17 features in total, so that the analysis of all possible variable combinations is too time intensive. On the other hand, we separately try models with all Twitter variables and all Yahoo Finance features, but those combinations are not able to beat the baseline approach. Hence, we can confirm the findings of [60] that too many features might reduce the predictive power.

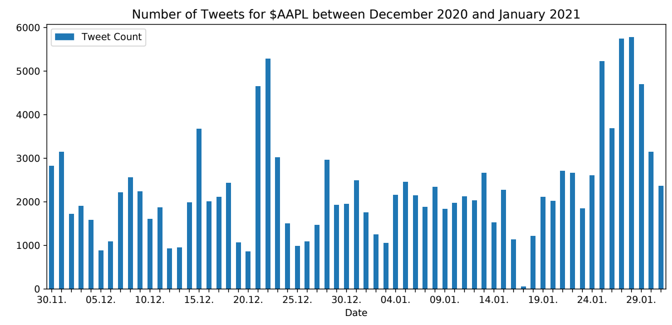

Figure 5. Number of Tweets per day for $AAPL.
Source: Own Representation.

The results for the Apple stock are illustrated in three different tables. The MSE summary for the data where the Twitter variables are only based on the ticker symbol can be seen in Table 1. Table 2 shows the results for the data set based on only the cashtag symbol and Table 3 for the combined Twitter data of ticker symbol and cashtag. Table 4 in the Appendix shows the used abbreviations for all features. In all settings, we are able to beat the baseline approach for both forecasting horizons. For the ticker symbol data the combination of lagged close price, trading volume, and average polarity per tweet leads to the best result for the shorter forecasting horizon. Contrary, for the `60m` period a more complex variable combination with lagged close price, number of positive tweets, average polarity per tweet, and volatility of number of tweets per minute minimizes the MSE. For the cashtag the results for the `30m` horizon suggest again a less complex variable combination. The best performing combination here were in addition to the lagged close price, the number of negative tweets, and the minimum number of tweets per minute. For the longer forecasting horizon the same variables as in the `30m` horizon plus the volatility of the number of tweets per minute gives the best performing model.

[‡]Note that on 17th January 2021 the streaming did not work correctly which is observable in both figures. However, it did not effect the analysis as it was a Sunday.

Table 1. MSE results for different feature combinations for the `AAPL` test data.
Source: Own Representation.

#	Forecast	Variables	MSE
Baseline	`30m`	3	0.00070
	`60m`	3	0.00104
1	`30m`	3,4	0.00068
	`60m`	3,13	0.00103
2	`30m`	3,4,11	0.00066
	`60m`	3,13,11	0.00097
3	`60m`	3,13,11,7	0.00097
4	`30m`	3, 5-16	0.00109
	`60m`	3, 5-16	0.00250
5	`30m`	0-4	0.00077
	`60m`	0-4	0.00114
6	`30m`	0-16	0.00083
	`60m`	0-16	0.00143

Table 2. MSE results for different feature combinations for the `$AAPL` test data.
Source: Own Representation.

#	Forecast	Variables	MSE
Baseline	`30m`	3	0.00069
	`60m`	3	0.00107
1	`30m`	3,14	0.00061
	`60m`	3,8	0.00098
2	`30m`	3,14,8	0.00059
	`60m`	3,8,14	0.00090
3	`60m`	3,8,14,7	0.00089
4	`30m`	3, 5-16	0.00069
	`60m`	3, 5-16	0.00098
5	`30m`	0-4	0.00078
	`60m`	0-4	0.00112
6	`30m`	0-16	0.00074
	`60m`	0-16	0.00099

Considering the combination of `AAPL` and `$AAPL` the results in Table 3 confirm the previous findings. On the one hand, the shorter horizon stops to improve faster. In fact, only adding feature for the number of negative tweets is valuable. On the other hand, the number of negative tweets, the number of positive tweets, and the average subjectivity per tweet improve the results of the `60m` forecasting horizon. It seems that even if the variable combinations for each approach are slightly different, there are features that lead repetitively to the best MSE. Therefore, it could be of interest to compare more stocks in future studies for figuring out if there are Twitter variables which are commonly and repeatable usable.

Comparing the best MSE for each of the three Apple results it seems like streaming Twitter data, which is directly connected to the financial discussion increases the predictive power. We find that for both forecast horizons the smallest MSE is obtained if we only use the `$AAPL` tweets.

Table 3. MSE results for different feature combinations for the combined `AAPL` and `$AAPL` test data.
Source: Own Representation.

#	Forecast	Variables	MSE
Baseline	`30m`	3	0.000679
	`60m`	3	0.001058
1	`30m`	3,14	0.000623
	`60m`	3,14	0.001029
2	`60m`	3,14,13	0.001021
3	`60m`	3,14,13,12	0.001003
4	`30m`	3, 5-16	0.000734
	`60m`	3, 5-16	0.001455
5	`30m`	0-4	0.000765
	`60m`	0-4	0.001131
6	`30m`	0-16	0.000794
	`60m`	0-16	0.001300

6. Conclusion

This paper provides a practical guide for stock price predictions with Twitter data by using a combination of a LSTM and sentiment classification. We offer a pathway to extend the modelling framework beyond common financial variables by using innovative variables based on Twitter data. On the one hand, we evaluate content unrelated characteristics of the data, such as the average number of tweets per minute. On the other hand, we generate sentiment variables, such as average polarity and subjectivity. We show in a case study for the stock prices of Apple, that these novel features improve the performance of our baseline LSTM model. Our study shows not only how to use innovative Twitter based variables for forecasting purposes, but also indicates its potential to improve the modelling capacity.

REFERENCES

1. A. A. Adebiyi, and C. K. Ayo, and M. O. Adebiyi, and S. O. Otokiti, *Stock price prediction using neural network with hybridized market indicators*, Journal of Emerging Trends in Computing and Information Sciences, vol. 3, no. 1, pp. 1–9, 2012.
2. A. A. Adebiyi, and A. O. Adewumi, and C. K. Ayo, *Comparison of ARIMA and artificial neural networks models for stock price prediction*, Journal of Applied Mathematics, vol. 2014, pp. 1–7, 2014.
3. G. V. Attigeri, and M. P. MM, and M. Radhika, and A. Nayak, *Stock market prediction: A big data approach*, TENCON 2015-2015 IEEE Region 10 Conference, pp. 1–5, 2015.
4. M. Baughman, and C. Haas, and R. Wolski, and I. Foster, and K. Chard, *Predicting Amazon Spot Prices with LSTM Networks*, in Proceedings of the 9th Workshop on Scientific Cloud Computing, Tempe, AZ, USA, pp. 1–7, 2018.
5. Y. Bengio, *Practical Recommendations for Gradient-Based Training of Deep Architectures*, in Neural Networks: Tricks of the Trade: Second Edition, edited by G. Montavon, and G. Orr, and K.-R. Müller, Springer-Verlag, Berlin, Heidelberg, pp. 437–478, 2012.
6. J. Bergstra, and Y. Bengio, *Random search for hyper-parameter optimization*, The Journal of Machine Learning Research, vol. 13, no. 1, pp. 281–305, 2012.
7. C. M. Bishop, *Pattern recognition and machine learning*, Springer-Verlag, New York, 2016.
8. J. Bollen, and H. Mao, and X.-J. Zeng, *Twitter mood predicts the stock market*, Journal of Computational Science, vol. 2, no. 1, pp. 1–8, 2011.
9. K. Chen, and Y. Zhou, and F. Dai, *A LSTM-based method for stock returns prediction: A case study of China stock market*, 2015 IEEE international conference on big data (big data), pp. 2823–2824, 2015.
10. R. Chen, and M. Lazer, *Sentiment analysis of twitter feeds for the prediction of stock market movement*, stanford edu Retrieved January, vol. 25, pp. 1–5, 2013.
11. S. Das, and R. K. Behera, and S. K. Rath,and others, *Real-time sentiment analysis of twitter streaming data for stock prediction*, Procedia computer science, vol. 132, pp. 956–964, 2018.
12. M. De Choudhury, and H. Sundaram,and A. John, and D. D. Seligmann, *Can Blog Communication Dynamics Be Correlated with Stock Market Activity?*, in Proceedings of the Nineteenth ACM Conference on Hypertext and Hypermedia, pp. 55—60, 2008.
13. L. Di Persio, and O. Honchar, *Artificial neural networks architectures for stock price prediction: Comparisons and applications*, International journal of circuits, systems and signal processing, vol. 10, no. 2016, pp. 403–413, 2016.
14. T. Fischer,and C. Krauss, *Deep learning with long short-term memory networks for financial market predictions*, European Journal of Operational Research, vol. 270, no. 2, pp. 654–669, 2018.
15. A.-C. Florea, and R.Andonie, *Weighted Random Search for Hyperparameter Optimization*, International Journal of Computers Communications & Control, vol. 14, no. 2, pp. 154—169, 2019.
16. Y. Gal, and Z. Ghahramani, *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks*, arXiv preprint arXiv:1512.05287, 2016.
17. A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*, O'Reilly Media, 2019.
18. I. Goodfellow, and Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.
19. A. Graves, and J. Schmidhuber, *Framewise phoneme classification with bidirectional LSTM and other neural network architectures*, Neural Networks, vol. 18, no. 5, pp. 602–610, 2005.
20. K. Greff, and R. K. Srivastava, and J. Koutník, and B. R. Steunebrink,and J. Schmidhuber, *LSTM: A search space odyssey*, IEEE transactions on neural networks and learning systems, vol. 28, no. 10, pp. 2222–2232, 2016.
21. A. Hasan, and S. Moin,and A. Karim, and S. Shamshirband, *Machine learning-based sentiment analysis for twitter accounts*, Mathematical and Computational Applications, vol. 23, no. 1, pp. 1–11, 2018.
22. M. Hiransha, and E. A. Gopalakrishnan, and V. K. Menon, and K. P. Soman, *NSE stock market prediction using deep-learning models*, Procedia computer science, vol. 132, pp. 1351–1362, 2018.
23. S. Hochreiter, and J. Schmidhuber, *LSTM can solve hard long time lag problems*, in Advances in neural information processing systems, pp. 473–479, 1997.
24. S. Hochreiter, and J. Schmidhuber, *A new model for stock price movements prediction using deep neural network*, in SoICT '17: Eighth International Symposium on Information and Communication Technology, pp. 57–62, 2017.
25. S. Jain, and R. Gupta,and A. A. Moghe, *Stock price prediction on daily stock data using deep neural networks*, in 2018 International Conference on Advanced Computation and Telecommunication (ICACAT), pp. 1–13, 2018.

26. G. Kant, and C. Weisser, and B. Säfken, *TTLocVis: A Twitter Topic Location Visualization Package*, Journal of Open Source Software, 5(54), 2507.
27. M. M. Khani, and S. Vahidnia, and A. Abbasi, *A Deep Learning Based Methods for Forecasting Gold Price with Respect to Pandemics*, 2020.
28. T. Kim, and H. Y. Kim, *Forecasting stock prices with a feature fusion LSTM-CNN model using different representations of the same data*, PloS one, vol. 14, no. 2, pp. 1–23, 2019.
29. D. P. Kingma, and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv preprint arXiv:1412.6980, 2017.
30. S. Loria, and P. Keen, and M. Honnibal, and R. Yankovsky, and D. Karesh, and E. Dempsey, and others, *Textblob: simplified text processing*, Secondary TextBlob: simplified text processing, vol. 3, 2014.
31. W. Ma, and Y. Wang, and N. Dong, *Study on stock price prediction based on BP neural network*, in 2010 IEEE International Conference on Emergency Management and Management Sciences, pp. 57–60, 2010.
32. P. Malhotra, and L. Vig, and G. Shroff, and P. Agarwal, *Long short term memory networks for anomaly detection in time series*, Proceedings, vol. 89, Presses universitaires de Louvain, pp. 89–94, 2015.
33. Y. Mao, and W. Wei,and B. Wang, and B. Liu, *Correlating S&P 500 stocks with Twitter data*, in Proceedings of the first ACM international workshop on hot topics on interdisciplinary social networks research, pp. 69–72, 2012.
34. D. Masters, and C. Luschi, *Revisiting Small Batch Training for Deep Neural Networks*, arXiv preprint arXiv:1804.07612, 2018.
35. S. Mehtab and J. Sen, *A robust predictive model for stock price prediction using deep learning and natural language processing*, Available at SSRN 3502624, 2019.
36. A. Mittal, and A. Goel, *Stock prediction using twitter sentiment analysis*, Standford University, CS229 (2011 http://cs229. stanford. edu/proj2011/GoelMittal-StockMarketPredictionUsingTwitterSentimentAnalysis.pdf), vol. 15, pp. 1–5, 2012.
37. M. Nikou, and G. Mansourfar, and J. Bagherzadeh, *Stock price prediction using DEEP learning algorithm and its comparison with machine learning algorithms*, Intelligent Systems in Accounting, Finance and Management, vol. 26, no. 6, pp. 164–174, 2019.
38. V. S. Pagolu, and K. N. Reddy and G. Panda; and B. Majhi, *Sentiment analysis of Twitter data for predicting stock market movements*, in 2016 international conference on signal processing, communication, power and embedded system (SCOPES), pp. 1345–1350, 2016.
39. P.-F. Pai, and C.-S. Lin, *A hybrid ARIMA and support vector machines model in stock price forecasting*, Omega, vol. 33, no. 6, pp. 497–505, 2005.
40. K. Pawar, and R. S. Jalem, and V. Tiwari, *Stock market price prediction using LSTM RNN*, in Emerging Trends in Expert Applications and Security, Springer, Singapore, pp. 493–503, 2019.
41. V. Pham, and T. Bluche, and C. Kermorvant, and J. Louradour, *Dropout improves Recurrent Neural Networks for Handwriting Recognition*, arXiv preprint arXiv:1312.4569, 2014.
42. R. Pimprikar, and S. Ramachandran, and K. Senthilkumar, *Use of machine learning algorithms and twitter sentiment analysis for stock market prediction*, International Journal of Pure and Applied Mathematics, vol. 115, no. 6, pp. 521-526, 2017.
43. A. Porshnev, and I. Redkin, and A. Shevchenko, *Machine learning in prediction of stock market indicators based on historical data and data from twitter sentiment analysis*, in 2013 IEEE 13th International Conference on Data Mining Workshops, pp. 440–444, 2013.
44. M. Qasem, and R. Thulasiram, and P. Thulasiram, *Twitter sentiment classification using machine learning techniques for stock markets*, in 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 834–840, 2015.
45. S. Raschka, and V. Mirjalili, *Python Machine Learning*, 3rd Ed., Packt Publishing, Birmingham, UK, 2019.
46. J. Roesslein, *tweepy Documentation*, Online] http://tweepy. readthedocs. io/en/v3, vol. 5, 2009.
47. M. Roondiwala, and H. Patel, and S. Varma, *Predicting stock prices using LSTM*, International Journal of Science and Research (IJSR), vol. 6, no. 4, pp. 1754–1756, 2017.
48. S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747, 2017.
49. B. Säfken, and A. Silbersdorff, and C. Weisser, *Learning deep: Perspectives on Deep Learning Algorithms and Artificial Intelligence*, Universitätsverlag Göttingen, Göttingen, 2020.
50. H. Sak, and A. Senior, and F. Beaufays, *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*, arXiv preprint arXiv:1402.1128, 2014.
51. L. Sayavong, and Z. Wu, and S. Chalita, *Research on Stock Price Prediction Method Based on Convolutional Neural Network*, in 2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS), pp. 173–176, 2019.
52. S. Selvin, and R. Vinayakumar, and E. A. Gopalakrishnan, and V. K. Menon, and K. P. Soman, *Stock price prediction using LSTM, RNN and CNN-sliding window model*, in 2017 international conference on advances in computing, communications and informatics (icacci), pp. 1643–1647, 2017.
53. J. Sen, and T. Datta Chaudhuri, *Stock price prediction using machine learning and deep learning frameworks*, in Proceedings of the 6th International Conference on Business Analytics and Intelligence, Bangalore, India, 2018.
54. D. Shah, and W. Campbell, and F. H. Zulkernine, *A comparative study of LSTM and DNN for stock market forecasting*, in 2018 IEEE International Conference on Big Data (Big Data), pp. 4148–4155, 2018.
55. M. Skuza, and A. Romanowski, *Sentiment analysis of Twitter data within big data distributed environment for stock prediction*, in 2015 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 1349–1354, 2015.
56. Y.-G. Song, and Y.-L. Zhou, and R.-J. Han, *Neural networks for stock price prediction*, arXiv preprint arXiv:1805.11317, 2018.
57. K. S. Vaisla,and A. K. Bhatt, *An analysis of the performance of artificial neural network technique for stock market forecasting*, International Journal on Computer Science and Engineering, vol. 2, no. 6, 2104–2109, 2010.
58. I. Vasilev, and D. Slater, and G. Spacagna, and P. Roelants,and V. Zocca, *Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow*, Packt Publishing Ltd, 2019.
59. T. T. Vu, and S. Chang, and Q. T. Ha, and N. Collier *An experiment in integrating sentiment features for tech stock prediction in twitter*, in Proceedings of the workshop on information extraction and entity analytics on social media data, pp. 23–38, 2012.

60. P. D. Yoo, and M. H. Kim, and T. Jan, *Machine learning techniques and use of event information for stock market prediction: A survey and evaluation*, in International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), pp. 835–841, 2005.
61. W. Zaremba, and I. Sutskever, and O. Vinyals, *Recurrent neural network regularization*, arXiv preprint arXiv:1409.2329, 2014.
62. T. Zhang, and S. Song, and S. Li, and L. Ma, and S. Pan, and L. Han, *Research on Gas Concentration Prediction Models Based on LSTM Multidimensional Time Series*, Energies, vol. 12, no. 1, pp. 1996-1073, 2019.
63. X. Zhang, and H. Fuehres, and P. A. Gloor, *Predicting stock market indicators through twitter "I hope it is not as bad as I fear"*, Procedia-Social and Behavioral Sciences, vol. 26, pp. 55–62, 2011.

Appendix

Table 4. Shortcuts for all variables.
Source: Own Representation.

#	Variable	Description
0	Open	Open Price
1	High	High Price
2	Low	Low Price
3	Close	Close Price
4	Volume	Volume
5	tw_count	Number of tweets
6	tw_mean	Average number of tweets per minute
7	tw_vola	Volatility of number of tweets per minute
8	tw_min	Minimum number of tweets per minute
9	tw_max	Maximum number of tweets per minute
10	tw_chars	Average number of characters per tweet
11	tw_pola	Average polarity per tweet
12	tw_subj	Average subjectivity per tweet
13	tw_n_pos	Number of positive tweets
14	tw_n_neg	Number of negative tweets
15	tw_ratio_pos	Ratio of positive tweets to total number of tweets
16	tw_ratio_neg	Ratio of negative tweets to total number of tweets

Listing 1: `TextBlob` example for polarity calculation.

```
In[2]: from textblob import TextBlob
  ...: good = TextBlob("good").sentiment.polarity
  ...: good
Out[2]: 0.7

In[3]: bad = TextBlob("bad").sentiment.polarity
  ...: bad
Out[3]: -0.6999999999999998

In[4]: good_and_bad = TextBlob("good and bad").sentiment.polarity
  ...: good_and_bad
Out[4]: 5.551115123125783e-17

In[5]: (good + bad) / 2 == good_and_bad
Out[5]: True

In[6]: not_good = TextBlob("not_good").sentiment.polarity
  ...: not_good
Out[6]: -0.35

In[7]: very_good = TextBlob("very_good").sentiment.polarity
  ...: very_good
Out[7]: 0.9099999999999999
```

Listing 2: `TextBlob` example for subjectivity calculation.

```
In [2]: from textblob import TextBlob
   ...: good = TextBlob("good").sentiment.subjectivity
   ...: good
Out[2]: 0.6000000000000001

In [3]: awesome = TextBlob("awesome").sentiment.subjectivity
   ...: awesome
Out[3]: 1.0

In [4]: awe_n_good = TextBlob("awesome and good").sentiment.subjectivity
   ...: awe_n_good
Out[4]: 0.8

In [5]: (good + awesome) / 2  == awe_n_good
Out[5]: True
```

Listing 3: Python Code

```
############################################
# Stock Data Stream
############################################
import yfinance as yf

# Chose your ticker
ticker_list = ["AAPL"]

path = ""
file_name = ""
start_date = "2020-11-25"        # first date for which tweets will be extracted
last_date = "2020-11-26"         # last date for which tweets will be extracted

for interval in ["1m", "5m", "15m", "30m", "1h"]:
    stock_data = yf.download(ticker_list,
                             start=start_date,
                             end=last_date,
                             interval=interval)
    stock_data.to_csv(path + file_name + "_" + start_date + "_" + last_date + "_" + interval + ".csv")


############################################
# Twitter Data Stream
############################################
import tweepy as tw
import pandas as pd
import datetime as dt

# Plug in developer account keys
consumer_key = ""
consumer_secret = ""
access_token = ""
access_token_secret = ""

# Give Python your developer account keys
auth = tw.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
# twitter api set ups
api = tw.API(auth,
             wait_on_rate_limit=True,
             wait_on_rate_limit_notify=True,
             retry_count=10000,
             retry_delay=60,
             timeout=120)

# Define variables below
path = ""                                         # folder where the data will be saved
start_date = "2021-01-13"                         # first day for extracting tweets
last_date = "2021-01-13"                          # last day (included) for extracting tweets

search_word_list = ["$AAPL", "AAPL"]

############################################
# Get Tweets
############################################

# search for each word in search list separately
for search_word in search_word_list:
    print("Current search word: " + search_word)

    # search for each date separately
    for date_since in pd.date_range(start=start_date, end=last_date):
        df = pd.DataFrame([])
        print(date_since.strftime('%Y-%m-%d'))
        # extract tweets
        tweets = tw.Cursor(api.search,
                           q=search_word,
                           extended=True,
                           lang="en",
                           until=(date_since + dt.timedelta(days=1)).strftime('%Y-%m-%d'),
                           since=date_since.strftime('%Y-%m-%d')).items()

        # put content of object into list
        users_locs = [[search_word,
                      tweet.user.screen_name,
                      tweet.user.location,
                      tweet.text,
                      tweet.created_at] for tweet in tweets]

        # make a df
        tweet_text = pd.DataFrame(data=users_locs,
                                  columns=["search_word", 'user', "location", "text", "date"])

        # merge the df with the other results for different dates and search words
        df = df.append(tweet_text)

        # save daily results
        df = df.reset_index(drop=True)
        df.to_csv(path + search_word + "_" + date_since.strftime('%Y-%m-%d') + ".csv")

############################################
#
# Functions
#
############################################

import pandas as pd
import datetime as dt
import numpy as np
from textblob import TextBlob
import re


# fct to concate daily twitter data to one df
def concat_twitter_data(path, search_word, start_date, last_date):
    results = pd.DataFrame([])
    for date in pd.date_range(start=start_date, end=last_date):
        results = results.append(pd.read_csv(path +
                                             search_word +
                                             "_" +
                                             date.strftime('%Y-%m-%d') + ".csv", index_col=0))

    return results.reset_index(drop=True)


# fct to formate the given date of the twitter data to the right format
def transform_twitter_time(twitter_data):
    results = twitter_data.copy()
    results["date"] = pd.to_datetime(results["date"], format='%Y-%m-%d %H:%M:%S')
    return results


# fct to transform the date of the stock data to time zone
def transform_stock_time(stock_data):
    results = stock_data.copy()
    results["date"] = results.index.tz_convert('Europe/Berlin')
    results["date"] = results["date"].dt.tz_localize(None)
    results = results.reset_index(drop=True)
    return results


# Cleaning the tweets
def cleanUpTweet(txt):
    # Remove mentions
    txt = re.sub(r'@[A-Za-z0-9_]+', '', txt)
    # Remove hashtags
    txt = re.sub(r'#', '', txt)
    # Remove retweets
    txt = re.sub(r'RT : ', '', txt)
    # Remove urls
    txt = re.sub(r'https?:\/\/[A-Za-z0-9\.\/]+', '', txt)
    return txt


# function to calculate subjectivity of tweets
def getTextSubjectivity(txt):
    return TextBlob(txt).sentiment.subjectivity

# function to calculate polarity of tweets
```

```
def getTextPolarity(txt):
    return TextBlob(txt).sentiment.polarity


# negative, nautral, positive analysis
def getTextAnalysis(a):
    if a < 0:
        return "Negative"
    elif a == 0:
        return "Neutral"
    else:
        return "Positive"


# fct to clean tweets and to calculate polarity, subjectivity and tweet_class
def prepare_twitter_variables(twitter_data):
    results = twitter_data.copy()
    results['clean_text'] = results['text'].apply(cleanUpTweet)
    results["polarity"] = results['clean_text'].apply(getTextPolarity)
    results["subjectivity"] = results['clean_text'].apply(getTextSubjectivity)
    results["tweet_class"] = results['polarity'].apply(getTextAnalysis)

    return results


# fct to merge stock and twitter data and create final variables
def add_twitter_variables(stock_data,
                          twitter_data,
                          tweet_lag=5,
                          subset_tweets_per=1):

    ########################################
    # Set ups
    ########################################
    # determine time interval of stock data (time diff (in seconds) / 60) = time diff in minutes
    # stock_t_int = stock_time_interval
    stock_t_int = int((stock_data["date"][1] - stock_data["date"][0]).seconds / 60)
    # copy stock data df and create new columns
    results = stock_data.copy()
    results["tw_count"] = "NA"              # number of tweets per stock_t_int
    results["tw_mean"] = "NA"               # mean of number of tweets per subset_tweets_per in stock_t_int
    results["tw_vola"] = "NA"               # volatility of number of tweets per subset_tweets_per in stock_t_int
    results["tw_min"] = "NA"                # min number of tweets per subset_tweets_per in stock_t_int
    results["tw_max"] = "NA"                # max number of tweets per subset_tweets_per in stock_t_int
    results["tw_chars"] = "NA"              # avg number of characters of tweets per stock_t_int
    results["tw_pola"] = "NA"               # avg polarity of tweets
    results["tw_subj"] = "NA"               # avg subjectivity of tweets
    results["tw_n_pos"] = "NA"              # number of positive tweets
    results["tw_n_neg"] = "NA"              # number of negative tweets
    results["tw_ratio_pos"] = "NA"          # share of positive tweets
    results["tw_ratio_neg"] = "NA"          # share of negative tweets

    # loop through all rows
    for i in range(0, stock_data.shape[0]):
        # get current time, which is the time of the Open Price
        # therefore, time interval of interest is Current Time + time interval of the data
        # as we analyse the clos price
        current_time = stock_data.iloc[i, :]["date"]
        # calculate time range
        cond_1 = twitter_data["date"] < (current_time + dt.timedelta(minutes=stock_t_int - tweet_lag))
        cond_2 = twitter_data["date"] >= (current_time - dt.timedelta(minutes=tweet_lag))
        # subset twitter data with tim range
        twitter_subset = twitter_data.loc[cond_1 & cond_2, :].copy().reset_index(drop=True)

        ########################################
        # Content unrelated variables
        ########################################
        # Compute Variable "tw_count"
        results.loc[i, "tw_count"] = twitter_subset.shape[0]

        # set ups for variables calculated afterwards
        tweets_per = pd.Series(np.zeros(stock_t_int))
        for x in np.arange(subset_tweets_per, stock_t_int + 1, subset_tweets_per):
            sub_cond_1 = twitter_subset["date"] >= (current_time + dt.timedelta(minutes=(int(x) - 1 - tweet_lag)))
            sub_cond_2 = twitter_subset["date"] < (current_time + dt.timedelta(minutes=(int(x) - tweet_lag)))
            tweets_per[x - 1] = twitter_subset.loc[sub_cond_1 & sub_cond_2, :].shape[0]

        # Compute Variables "tw_mean", "tw_vola", "tw_min", "tw_max"
        results.loc[i, "tw_mean"] = tweets_per.mean()
        results.loc[i, "tw_vola"] = tweets_per.var()
        results.loc[i, "tw_min"] = tweets_per.min()
        results.loc[i, "tw_max"] = tweets_per.max()

        # Compute Variable "tw_chars"
        twitter_subset["tw_chars"] = twitter_subset.text.apply(len)
        results.loc[i, "tw_chars"] = twitter_subset["tw_chars"].mean()
        ########################################
        # Content related variables
        ########################################
        results.loc[i, "tw_pola"] = twitter_subset["polarity"].mean()
        results.loc[i, "tw_subj"] = twitter_subset["subjectivity"].mean()
        results.loc[i, "tw_n_pos"] = twitter_subset.loc[twitter_subset.tweet_class == "Positive", :].shape[0]
        results.loc[i, "tw_n_neg"] = twitter_subset.loc[twitter_subset.tweet_class == "Negative", :].shape[0]
        results.loc[i, "tw_ratio_pos"] = results.loc[i, "tw_n_pos"] / twitter_subset.shape[0]
        results.loc[i, "tw_ratio_neg"] = results.loc[i, "tw_n_neg"] / twitter_subset.shape[0]

    return results

####################################################
#
# Model
#
####################################################

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
from kerastuner.tuners import RandomSearch
import Functions as fct
import yfinance as yf
import keras
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


####################################################
#
# Prepare Data Set
#
####################################################

# loop to run RandomSearch for selected variable combinations
for z in [0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]:

    subset_col = [3, z]

    data_training = train_data.iloc[:, subset_col].copy()
    data_test = test_data.iloc[:, subset_col].copy()

    scaler = MinMaxScaler()
    data_training = scaler.fit_transform(data_training)

    X_train = []
    y_train = []
    step_size = 1

    ############################################################################################################
    # Forecasting Horizon one period (e.g. if time interval of data is 30m, then we make a 30m prediction)
    ############################################################################################################
    if horizon == 1:

        for i in range(step_size, data_training.shape[0]):
            X_train.append(data_training[i - step_size:i])
            y_train.append(data_training[i, 0])

        X_train, y_train = np.array(X_train), np.array(y_train)
        data_training = train_data.iloc[:, subset_col].copy()
        past_step_days = data_training.tail(step_size)
        df = past_step_days.append(data_test,
                                   ignore_index=True)
        inputs = scaler.transform(df)
        X_test = []
        y_test = []

        for i in range(step_size, inputs.shape[0]):
            X_test.append(inputs[i-step_size:i])
            y_test.append(inputs[i, 0])

        X_test, y_test = np.array(X_test), np.array(y_test)

    ############################################################################################################
    # Forecasting Horizon two periods (e.g. if time interval of data is 30m, then we make a 60m prediction)
    ############################################################################################################
    elif horizon == 2:
        for i in range(step_size, data_training.shape[0] - 1):
            X_train.append(data_training[i - step_size:i])
            y_train.append(data_training[i + 1, 0])

        X_train, y_train = np.array(X_train), np.array(y_train)
        data_training = train_data.iloc[:, subset_col].copy()
        past_step_days = data_training.tail(step_size)
        df = past_step_days.append(data_test,
                                   ignore_index=True)

        inputs = scaler.transform(df)
        X_test = []
        y_test = []

        for i in range(step_size, inputs.shape[0] − 1):
            X_test.append(inputs[i−step_size:i])
            y_test.append(inputs[i + 1, 0])

        X_test, y_test = np.array(X_test), np.array(y_test)
    else:
        print("You did not specify horizon correctly.")

    ####################################################
    #
    # Create Model
    #
    ####################################################

    def build_model(hp):
        model = Sequential()

        # number of hidden layer
        hidden = hp.Int("n_hidden",
                        min_value=0,
                        max_value=3)

        # first LSTM−Layer
        model.add(LSTM(units=hp.Int("n_units1",
                                    min_value=17,
                                    max_value=500,
                                    step=50),
                       activation=hp.Choice("v_activation",
                                            values=["relu", "tanh", "sigmoid"],
                                            default="relu"),
                       input_shape=(X_train.shape[1], X_train.shape[2]),
                       return_sequences=True if hidden > 0 else False))

        # add hidden layers (depends on before generated number for hidden)
        if hidden > 0:
            for layer in range(hidden):
                model.add(Dropout(hp.Float("v_dropout_hidden" + str(layer + 1),
                                           min_value=0.05,
                                           max_value=0.95,
                                           step=0.05)))

                model.add(LSTM(units=hp.Int("n_units_hidden" + str(layer + 1),
                                            min_value=17,
                                            max_value=250,
                                            step=50),
                               activation='relu',
                               return_sequences=True if layer != hidden else False))

        model.add(Dropout(hp.Float("v_dropout",
                                   min_value=0.05,
                                   max_value=0.95,
                                   step=0.05)))
        model.add(Dense(units=1))

        model.compile(optimizer="adam",
                      loss='mean_squared_error')

        return model


    ####################################################
    #
    # Search For Best Model
    #
    ####################################################
    tuner = RandomSearch(build_model,
                         objective='val_loss',
                         max_trials=50,
                         seed=1,
                         executions_per_trial=3,
                         directory="",
                         project_name="")
    early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
                                                      restore_best_weights=True)

    tuner.search(X_train,
                 y_train,
                 epochs=1500,
                 batch_size=32,
                 validation_split=0.2,
                 callbacks=[early_stopping_cb],
                 verbose=1)
    # Retrieve the best model.
    best_model = tuner.get_best_models(num_models=1)[0]

    # save best model for that variable combination
    best_model.save("best_model.h5")

    # Evaluate the best model with test data
    loss = best_model.evaluate(X_test, y_test)
```