universität wien

MAGISTERARBEIT / MASTER'S THESIS

Titel der Magisterarbeit / Title of the Master's Thesis

„Application of machine learning algorithms to stock price movement"

verfasst von / submitted by

Patrycja Pultowicz, B.B.A. BSc MA MSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Magister der Sozial- und Wirtschaftswissenschaften (Mag.rer.soc.oec)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 951
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Magisterstudium Statistik
Betreut von / Supervisor:	ao. Univ.-Prof. Dipl.-Ing. Dr. Erhard Reschenhofer

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Machine learning algorithms like neural networks and gradient boosting have significantly gained in popularity in the last five years in the areas of speech recognition (Chan et al., 2016), image classification (Xie et al., 2017), computer vision (Howard et al., 2017), natural language processing (Gu et al., 2018), medical image analysis (Esteva et al., 2017) as well as neural information processing (Ke et al., 2017). Deep neural networks are, thereby, amongst the most widely used supervised learning algorithms for many artificial intelligence applications as they show to achieve state-of-the-art accurate results on large and complex data sets (Sze et al., 2017). Due to their strong prediction performance, neural network algorithms have recently also gained large attention in the fields of financial market prediction (Long et al., 2019; Chong et al., 2017; Nelson et al., 2017).

This paper examines the prediction accuracy of machine learning algorithms for financial market data, motivated by the work of Fischer and Krauss (2018) on deep learning with long short-term memory (LSTM) networks. In their work, the authors compare the prediction quality of LSTM networks on directional movements of S&P 500 stocks from 1992 until 2015 to the memory-free classification methods random forest (RF), deep neural net (DNN) and logistic regression classification (LG). According to the authors' findings, LSTM networks outperform memory-free classification models. The prediction outperformance was particularly good in the first period of their long-term data sample (1992-2009).

In our empirical study, we apply four machine learning algorithms for predicting directional price movement in a binary classification problem on a recent data set of S&P 500 stocks, from 2017 until 2019. We consider two target settings for classifying the daily return into class {0,1}: first, the return is classified as 1 if it is above the median return of all stocks and 0 else; second, the return is classified as 1 if the return is positive and 0 else. The model outputs are predicted target probabilities which are transformed into predicted classes {0,1} and compared to the observed classes. Our experimental results show that none of the machine learning algorithms outperform a purely random classification of a coin flip in terms of prediction accuracy.

In the first part of the paper, we introduce the models used in the empirical application, that are logistic regression, random forest, standard deep feedforward neural networks and LSTM networks. In the second part, we describe the data and methodology, followed by analysing the results. In the last section of the paper, we discuss our findings.

2. MACHINE LEARNING ALGORITHMS

This chapter aims at providing a broad understanding of the prediction models that are applied in this paper. We start by introducing the general notation that is used throughout this chapter.

We consider the following problem: We have a binary response variable y and real-valued features x of dimension d. We set $f(x)$ to be the true probability of y being equal to 1 under input x, and $1 - f(x)$ being equal to 0 under input x:

$$p(y = 1|\, x) = f(x)$$
$$p(y = 0|\, x) = 1 - f(x)$$

We have a data set of n pairs of observations of (x, y) which we denote by D_n. During this chapter, we describe several **supervised learning algorithms** that provide a function $\hat{f}$ which only depends on the data set D_n, such that $\hat{f}(x)$ approximates the true probability function $f(x)$. Supervised learning algorithms are, in contrast to unsupervised learning algorithms, algorithms that learn a function $\hat{f}(x)$ based on n pairs of observations (x, y). On the other hand, unsupervised learning algorithms aim at extracting information (features) from the underlying distribution of x only and do not require a target y (Goodfellow et al., 2016: Chapter 5).

We only consider binary regression models (class $K = 2$), instead of the general K-class regression problem.

The function $\hat{f}$ is the minimizer of

$$-\sum_{i=1}^{n}(1 - y_i) * \log\left(1 - \hat{f}(x_i)\right) + y_i * \log\left(\hat{f}(x_i)\right)$$

within a certain class of functions that is defined by the learning algorithm. The loss function is called the **binary cross-entropy loss**.

We first introduce artificial neural networks including feedforward neural networks and recurrent neural networks. As a specialization of recurrent neural networks, we additionally examine the structure of LSTM networks. In the second part of this chapter, we describe binary logistic regression which is used as the baseline model in the analytical part of this paper, followed by a brief introduction to random forests.

1.1. ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANN) are regression or classification models in which interconnected node structures map input values x (such as an image) to an output y in an affine transformation function through multiple levels of layers. The nodes in a neural network are called **neurons** that are small processing units joined by weighted connections and arranged into **layers**: an input layer x, one or several hidden layers h representing the **depth** of the network, and an output layer y. The network is activated by an input to some or all the nodes, and the information processing flows throughout the network along the weighted connections to the output layer (Hastie et al., 2008: Chapter 11; Graves, 2012: Chapter 3).

Various forms and properties of artificial neural networks have been developed and studied over the years, of which the most common differentiation is between ANN with cyclic connections, also referred to as recursive or **recurrent neural networks**, and with acyclic connections, so-called **feedforward neural networks** (Graves, 2012: Chapter 3).

The simplest form of an artificial neural network is a neural network with one hidden layer. Networks with more than one hidden layer are referred to as **deep neural networks** or **multilayer perceptrons (MLPs)** (Goodfellow et al., 2016: Chapter 6), whereby the latter are often referred to deep feedforward neural networks only. Deep networks have shown to work better for complex data, but generally also yield competitive prediction results for various forms of data complexity compared to a simple neural network with one hidden layer, according to literature (Deng et al., 2013; Ciresan et al., 2012).

Artificial neural networks became state-of-the-art methods in the fields of artificial intelligence (AI) and biomedicine in the last ten years motivated by technological advancements in computer science, the increasing digitization of society and, with it, the availability of large datasets (also referred to as *big data*). The continuous growth in model size due to faster computers and larger memory benefits the possibility to analyse an expanding depth of artificial neural networks for further scientific and technological progresses in all sorts of fields. The idea driving the development of neural networks is the belief that deep neural networks can succeed in complex analytical tasks similarly to a human brain (Goodfellow et al., 2016).

In this chapter and its subchapters, we follow the definitions of the book *Deep Learning* by Goodfellow et al. (2016) and therefore do not cite it in the following sections. Moreover, we refer to several papers that provide more detailed information and are cited at the corresponding paragraphs.

The following graphs below demonstrate two broad classes of neural networks. We describe the networks in more detail in the subchapters thereafter:

- a directed graph without cyclic connections – a so-called **feedforward neural network (FNN)** (**Figure 1**: with one hidden layer; **Figure 2**: a deep neural network with two hidden layers)
- a directed graph with cyclic connections– a so-called **recurrent neural network (RNN)** (**Figure 3**). In contrast to a feedforward neural network, a recurrent neural network processes the information from both, the current input parameter (at time t) and the previous hidden layer state (at time $t-1$). Thus, the hidden state is passed forward through time

Figure 1: *A graphical representation of a* ***feedforward neural network*** *with a hidden single layer and one neuron at the output layer based on a binary regression problem (adopted from Goodfellow et al., 2016: Fig.6.2);* y *refers to a class value of* 0 *or* 1*. In this illustration the input vector is* 3×1 *dimensional, the respective weights matrix is* 5×3 *and the output is the class parameter*

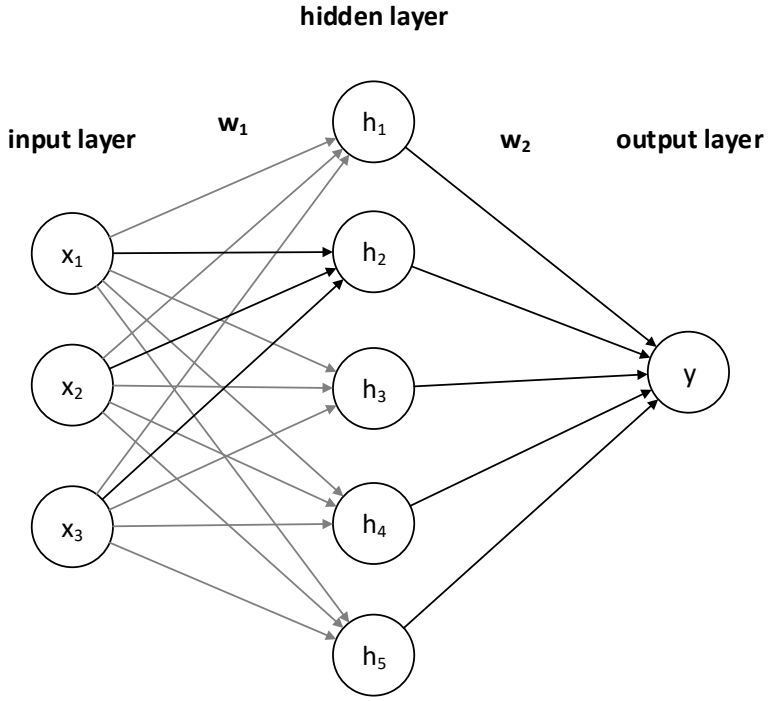


Figure 2: *A graphical representation of a feedforward neural network with two hidden layers, from the class of* ***deep neural networks*** *(reproduced from Nielsen (2015): online). In this illustration the input vector* x *is* 3×1*-dimensional and the respective weights matrix* w_1 *in the first layer has dimension* 5×3*. The first layer transformation results in a* 5×1 *hidden vector* $h^{(1)}$*. The weights matrix in the second hidden layer,* w_2*, is* 4×5*, resulting in a new hidden vector* $h^{(2)}$ *of dim* 4×1*. The output layer is ultimately mapped to the class parameter* y *using a new weights matrix* w_3 *of dim* 1×4

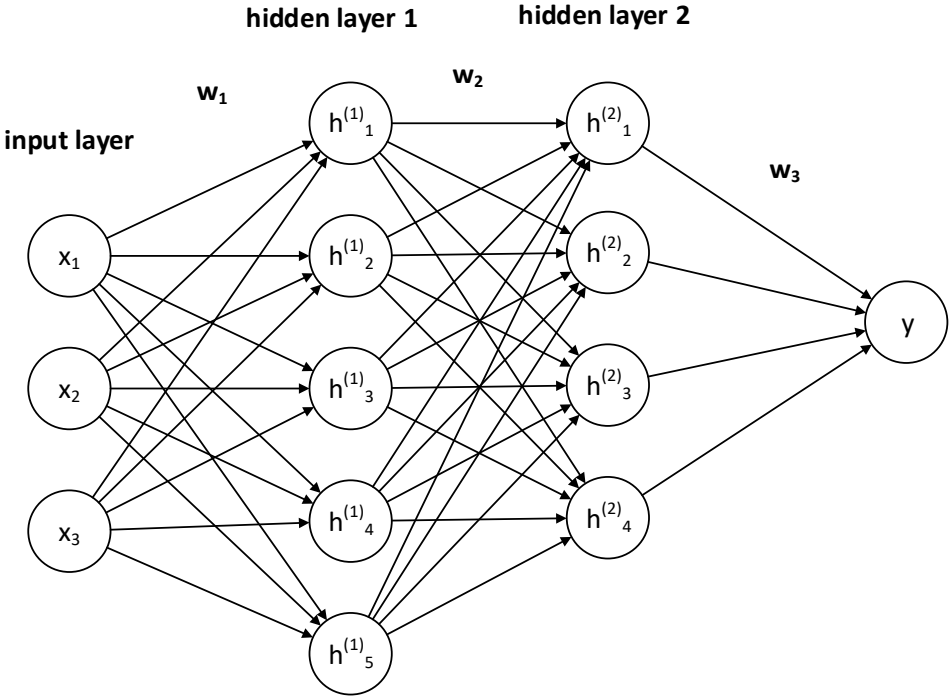
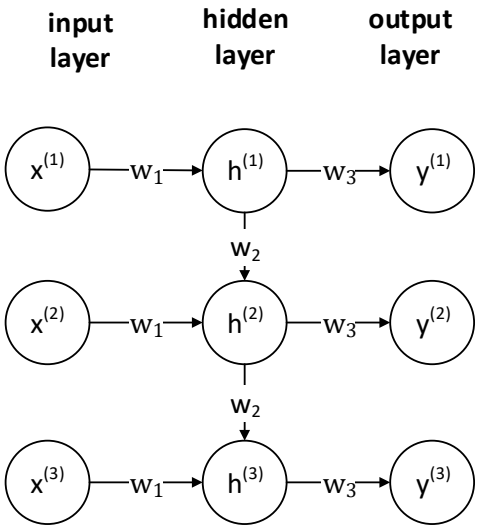


Figure 3: *A graphical representation of a* ***recurrent neural network*** *(RNN) with one cyclic connection in each node of the hidden layer (adopted from Goodfellow et al., 2016: Fig.10.2).* $x^{(t)}$ *describes the* a*-dimensional feature at time* t*,* $h^{(t)}$ *describes the* b*-dimensional hidden layer at time* t *and* $y^{(t)}$ *is our target at time* t*. The weights matrix* w_1 *has dimension* $b \times a$*,* w_2 *has dimension* $b \times b$ *and* w_3 *has dimension* $1 \times b$



The transformation steps from x to y are referred to as **forward-propagation**. During training, forward propagation computes the loss function as described at the beginning of this section, which is often referred to as scalar cost. The reverse information flow from the cost back through the network is called **back-propagation**. The goal of backpropagation is to find the optimal weights of the network. Backpropagation applies the chain rule to compute the gradients of the loss function with respect to the weights. The weights are iteratively updated by stochastic gradient descent steps until the loss function reaches a local minimum.

Besides the neural network schemes mentioned above, deep neural networks can comprise various other forms of networks, of which the most common ones are *convolutional networks*. Convolutional networks get their name by employing *convolution* instead of general matrix multiplication in at least one of their layers. They are used for data with a grid-like topology such as image data as 2D grid of pixels or time-series data as 1D grid data. Convolutional networks are popularly applied in computer vision and pattern recognition (Gu et al., 2018).

1.1.1. FEEDFORWARD NEURAL NETWORKS

Feedforward neural networks are characterized by an information flow without cyclic connections from a layer output back to itself. Hence, in a feedforward neural network, information flows through each layer starting from the input x to the output class y as illustrated in **Figure 1** and **Figure 2**. .

In the following paragraphs, we describe a feedforward neural network with one hidden layer. The *input layer* is our feature vector x. The first layer, thereafter, is called *hidden layer*. The hidden layer depends on x, a weight matrix w_1, a bias vector b_1 and a fixed, nonlinear-activation function $g^{(1)}$. We write the hidden layer as

$$h = g^{(1)}(x; w_1, b_1) = g^{(1)}(w_1 x + b_1)$$

The activation function $g^{(1)}$, which is used to train the features in the hidden layer, is a one-dimensional function that is applied component-wise. The values of the hidden layer h are then used as the inputs for the next layer of the network which is the output layer. The output layer depends on h, a weight matrix w_2, a bias vector b_2 and a nonlinear-activation function $g^{(2)}$. We write the output layer as

$$y = g^{(2)}(h; w_2, b_2) = g^{(2)}(w_2 h + b_2)$$

Again, $g^{(2)}$ is a one-dimensional function that is applied component-wise. In a neural network with more than one hidden layer, we have the same nested structure as before and only the last hidden layer is used as input for the output layers.

The complete feedforward network can be written as follows:

$$y = g^{(2)}(g^{(1)}(x; w_1, b_1); w_2, b_2)$$

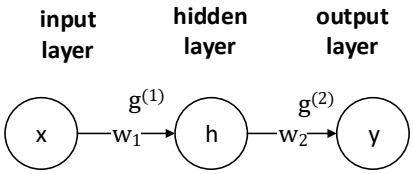
A commonly used activation function in deep learning models is the **Rectified Linear Unit** (ReLU), which is defined by $g(z) = \max(0, z)$.

Hence, if $g^{(2)}$ is a linear output function and $g^{(1)}$ is the nonlinear ReLU activation function, the network can be written as follows:

$$y = w_2 \max(0, w_1 x + b_1) + b_2$$

Figure 4 illustrates the logical scheme of such a simple neural network.

Figure 4: *Logical node structure of a simple neural network with one hidden layer, where x is the input vector from the input layer, w_1 is the weights matrix for x used in the mapping from the input to the hidden layer h, h and its weight composition w_2 is then mapped to an output parameter y (Goodfellow et al., 2016: Fig.6.2.)*



Similarly, a network with two hidden layers is defined by

$\hat{f}(x) = g^{(3)}\big(g^{(2)}\big(g^{(1)}(x; w_1, b_1); w_2, b_2\big); w_3, b_3\big)$, where x is the input variable, and w_i and b_i for $i \in \{1,2,3\}$ are the parameters of the neural network. The optimal parameters are obtained by backpropagation such that they approximately minimise the loss function to approximate some classifier function f

1.1.2. RECURRENT NEURAL NETWORKS

Artificial neural networks whose connections form cycles are called *recurrent neural networks (RNN)* (Graves, 2012: Chapter 3).

While a feedforward neural network can only map from an input to an output vector, RNNs can basically map recurrently throughout any previous input to an output and thus store representations of input events in form of activations within the network's internal state (Graves, 2012: Chapter 3; Hochreiter and Schmidhuber, 1997).

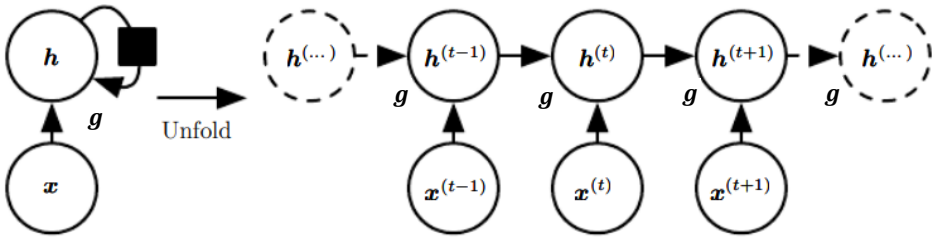
A simple example form of a recurrent neural network is given by

$$h^{(t)} = g(h^{(t-1)}, x^{(t)}; w_t, b_t),$$

where $h^{(t)}$ is the (hidden) state of the system at time t which refers back to the state at time $t-1$, and $x^{(t)}$ is an input signal that is incorporated into the state h at time t.

For a finite number of time steps T, the network can be computed by recursively applying the definition with the same set of weights for $T-1$ times. This process can be described as **unfolding** a network into a computational graph with a repetitive structure of events, which is exemplarily shown in **Figure 5**:

Figure 5: *A recurrent neural network with no outputs and a recurrent connection between the hidden states. The Illustration on the left is the folded network, the black square indicates a time delay of a single time step. The illustration on the right shows the unfolded network where each state is a time instance, in which x is processed into a hidden state h that is passed forward in time (derived from Goodfellow et al., 2016: Fig.10.2)*



Recurrent neural networks can be designed in a wide variety of ways, of which some of the common ones include the following patterns:

- Recurrent connections exist between hidden states only, and at each state a prediction output is produced. An input $x^{(t)}$ and the information from the previous hidden state $h^{(t-1)}$ is mapped to an output $o^{(t)}$. These output values are then compared to a corresponding training target $y^{(t)}$ using a loss function $L^{(t)}$. At each state transition (that is, input-to-hidden, hidden-to-hidden, hidden-to-output), some weight matrix is used. An illustration is provided by Goodfellow et al. (2016), Fig.10.3.
- Recurrent connections exist between an output at one time step and a hidden state at the next time step. Thereby, at each time step t, the input $x^{(t)}$ and the output sequence from the previous state $o^{(t-1)}$ is processed in the hidden layer activation function $h^{(t)}$ that is mapped to an output sequence at time t, $o^{(t)}$. In contrast to the previous network pattern, this RNN form is trained towards a specific prediction value o that is the only

information send to the next state. As the previous state h is only indirectly connected to the present via its prediction output, the network can be easier to parallelize and thus to train. An illustration is provided by Goodfellow et al. (2016), Fig.10.4.

- Recurrent connections exist between the hidden states, but there is a single network output o^T containing the summary of the sequence's history. This network has the advantage that it produces a fixed-size representation for further processing or target comparison. An illustration is provided by Goodfellow et al. (2016), Fig.10.5.
- Recurrent connections exist between the hidden states and information is processed both forward and backward in time, referred to as **bidirectional recurrent neural networks**. In such networks, an input $x^{(t)}$ is processed in a hidden forward and backward layer $h_f^{(t)}$ resp. $h_b^{(t)}$. The information from those hidden layers at time t are mapped to an output $o^{(t)}$, which is compared to a corresponding training target $y^{(t)}$, and are passed forward resp. backward in time. An illustration is provided by Graves (2012), Fig.3.5. This network type is in several ways different to a standard RNN. First, a standard RNN does not consider future states. Second, in a bidirectional network, there are no direct connections between the forward and backward states, which makes the unfolded graph in contrast to a standard RNN **acyclic**.

1.1.3. LONG SHORT-TERM MEMORY NETWORKS

A long short-term memory (LSTM) network is a special form of a **recurrent neural network** which uses *memory cells*, each with a self-recurrent *storage cell* responsible for storing values over some time interval and three internal information processing states: the input gate, the output gate and the forget gate. A memory cell, thereby, describes a state of information flow at time t, and the cells are connected recurrently to the rest of the network. A detailed illustration of an LSTM network cell is given in **Figure 6** (Hochreiter and Schmidhuber, 1997).

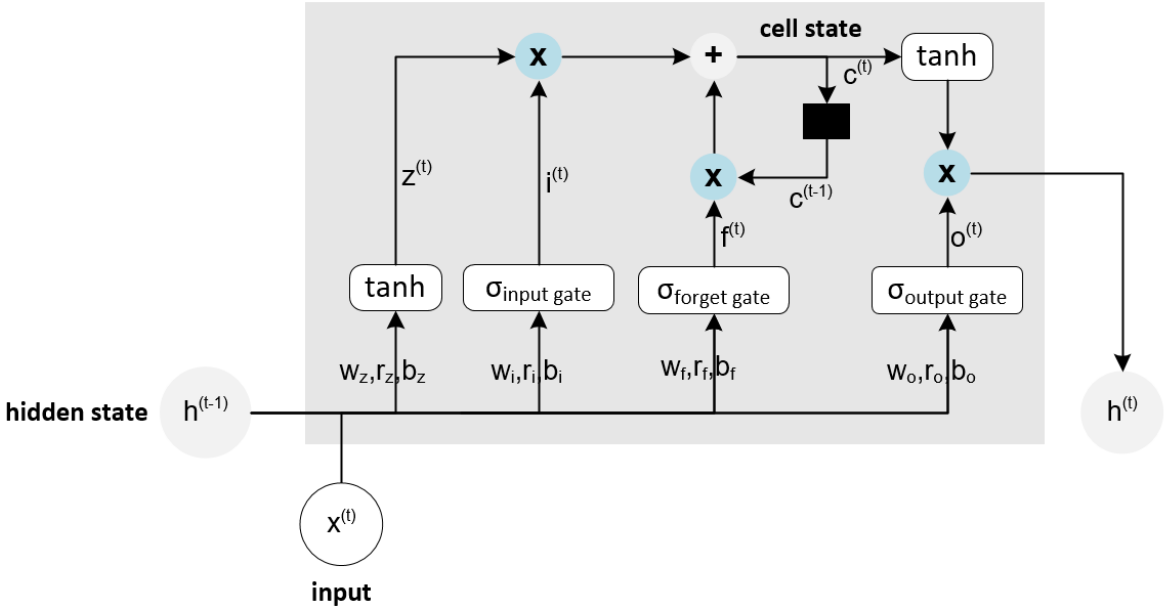
LSTM was initially proposed by Sepp Hochreiter and Jürgen Schmidhuber and included only the output gate, the cell state and the input gate. The idea of LSTM was largely motivated by the fact that the prediction error rate vanishes or blows up (depending on the input values) in the process of flowing through the recurrent connections in backpropagation. The goal of an LSTM network is, thus, to use constant error flows in both the output gate and the previous states to keep the error rate in backpropagation constant. This approach is also referred to as **constant error carrousel** (CEC) (Hochreiter, 1991).

The three gates, input gate, output gate and forget gate, are nonlinear summation units for collecting the input from outside and inside the LSTM cell, each with a gate activation function that is typically a logistic sigmoid function ($\sigma(y) = \frac{e^y}{1+e^y}$, i.e. gate activation is between 0 (gate closed) and 1 (gate open)). The block's output to be processed further in the network is generated by the output gate multiplication (Graves, 2012: Chapter 3).

The LSTM network works as follows:

New information given by $x^{(t)}$ and a previous memory cell's output $h^{(t-1)}$ enters the memory cell through $z^{(t)}$. Whether this information is further processed in the memory cell is, however, determined by the value of the **input gate** $i^{(t)}$ which is influenced by x_t and h_{t-1}, evaluated in the (nonlinear) sigmoid activation function. As such, the input gate protects the memory content going into the network cell from irrelevant information by blocking or letting pass the cell input $z^{(t)}$ through its gate function. The **cell state** unit, $c^{(t)}$, is a linear self-recurrent connection with a delay-feedback of 1 time step whose weight is controlled by the forget gate $f^{(t)}$, which sets its weights to a value between 0 and 1 based on the sigmoid activation function. The cell state builds the basis of the **constant error carrousel** and, in some approaches, can also serve as an extra input to the other gates. While the input and output gates multiply the input resp. the output of the cell, the **forget gate** multiplies the previous states of the cell. If the input gate is closed while the forget gate is open, the memory cell processes the memory of the first input only. Differently put, the forget gate provides a way for the memory cell to 'forget' a previous input and, hence, remove information that are no longer necessary (***cell reset***). Therefore, the forget gate is a crucial state in optimizing the network's performance. The output of the cell is selected or can be blocked to be passed forward to the next memory cell by the sigmoid activation function of the **output gate** $o^{(t)}$. Typically, the connections to the gates are differently weighted and can have distinct biases, denoted in **Figure 6** as $w^{(.)}, r^{(.)}$ and $b^{(.)}$ (Hochreiter and Schmidhuber, 1997; Graves, 2012: Chapter 4; Goodfellow et al., 2016: Chapter 10).

Figure 6: *Structure of an LSTM network with an input, forget and output state, each with a sigmoid gate activation function, a cell input for the input activation function* $tanh$ *(with weights* w_z *and bias* b_z*), a single memory cell state (for the constant error carousel) and an* $tanh$ *output activation function. The output of the memory cell is recurrently flowing to the next memory cell (reproduced from Greff K. et al., 2017: Fig.1)*



Formally, an LSTM network cell structure can be described as follows (notations are derived from Greff K. et al., 2017):

$z^{(t)} = g\left(w_z x^{(t)} + r_z h^{(t-1)} + b_z\right)$ computes the memory cell's input gate $z^{(t)}$ with input weight matrix w_z, some bias vector b_z and some recurrent weight matrix r_z on a previous memory cell's output vector $h^{(t-1)}$[1]. For the component-wise non-linear input activation function g a *hyperbolic tangent* function is typically used.

$i^{(t)} = \sigma\left(w_i x^{(t)} + r_i h^{(t-1)} + b_i\right)$ defines the input gate with a logistic sigmoid activation function σ, weight matrices w_i and r_i for the input vector $x^{(t)}$ resp. previous hidden state $h^{(t-1)}$, and some input bias vector b_i.

$c^{(t)} = z^{(t)} \odot i^{(t)} + f^{(t)} \odot c^{(t-1)}$ defines the cell state. The symbol $\odot$ denotes the component-wise multiplication of two vectors.

[1] Other inputs for the input gate can also be outputs from other memory cells, $y^{(t-1)}$, additional or instead of a previous hidden layer's output $h^{(t-1)}$

$f^{(t)} = \sigma\left(w_f x^{(t)} + r_f h^{(t-1)} + b_f\right)$ defines the forget gate with a logistic sigmoid activation function σ, weight matrices w_f and r_f, and some input bias b_f.

$o^{(t)} = \sigma\left(w_o x^{(t)} + r_o h^{(t-1)} + b_o\right)$ defines the output gate with a logistic sigmoid output activation function σ, weight matrices w_o and r_o, and some input bias b_o.

Finally, the memory cell's output is given by: $h^{(t)} = g(c^{(t)}) \odot o^{(t)}$, whereby g is a component-wise non-linear output activation function which is often also defined as a *hyperbolic tangent* function.

Using backpropagation, the gradients for the weights are calculated based on the loss function (in our example, the entropy as described at the beginning of this chapter) to optimize the network's prediction performance. Thereby, the derivative of each gate is passed down for training from the layer above (Goodfellow et al., 2016: Chapter 10).

Several variant forms of LSTM networks have since been developed, of which the most commonly known include *peephole connections* (in which the cell state is connected to all the gates in order to control the gates' timings, and the output activation function was removed), and *full gradient* network (which employs full backpropagation through time (BPTT)) (Greff et al., 2017).

1.2. BINARY LOGISTIC REGRESSION

Binary logistic regression is a 2-class linear classification approach which models the probability of y being in class $\{0,1\}$ via linear functions in the input x. By using the logistic sigmoid function, the output of the linear function is transformed to a probability value for a given class to be 0 respectively 1 (Goodfellow et al., 2016: Chapter 5).

In the following sections, we follow the definitions from the book *Deep Learning* by Goodfellow et al. (2016) and therefore do not cite it in this chapter.

Let us consider a linear model with a feature vector x, a weights vector w and a binary response variable y. The probability for y to be in class $\{0,1\}$ under the weights' parameter w is denoted as:

$$p(y = 1|x; w) = \sigma(w^T x) = \frac{\exp(w^T x)}{1+\exp(\theta^T x)}$$
$$p(y = 0|x; w) = 1 - \sigma(w^T x)$$

By using maximum likelihood estimation on the data set D_n, we find the best parameter vector (weights) for the probability distribution $p(y|x;w)$. This can be done by minimizing the negative log-likelihood using gradient descent.

Any interaction between the input variables are not taken into consideration in this model as it would be possible in the hidden layer(s) of a neural network, which is why logistic regression is limited to simple classification problems. A form of statistical linear model is though commonly used as a default baseline in many comparison applications and it can perform well, depending on the complexity of the underlying problem.

1.3. RANDOM FOREST

Random forest is a modification of *bagging* or *bootstrap aggregation* which produces multiple decision trees from random samples of the training set D_n and fits a model on this subset based on randomly selected variables. In random forest, a classification or regression tree is used for fitting. The random forest predictor is computed as the average of all its trees (Hastie et al., 2008: Chapter 15)

In this subchapter, we follow the definitions of the book *The Elements of Statistical Learning* (Chapter 15) by Hastie et al. (2016) and therefore do not cite it in the following sections.

In random forest, the goal is to fit many de-correlated identically distributed trees. De-correlation is created by randomly sampling subsets of the training data D_n and randomly sampling $m \leq d$ of the d input features as candidates for splitting. For each tree (also referred to as *decision tree*), the learning algorithm partitions the input space of x into two non-overlapping subspaces.

The optimal split is determined by comparing the loss function (binary cross-entropy loss) for each possible split in one of the features. The best split-point is chosen that minimizes the loss function. This partitioning step is repeated for each terminal node of the tree until the minimum node size n_{min} is reached.

Formally, the kth tree provides a classifier $T_k(x)$ that predicts y associated with input x based on the sample data, whereby T_k is a tree-based function as described before. After K such trees are grown, the random forest predictor is computed as

$$\hat{f}(x) = \frac{1}{K}\sum_{k=1}^{K} T_k(x)$$

The result can be compared to a majority vote for a class; as each tree (classifier) casts a vote for the most popular class for an input x, the class with the most votes wins (Breiman, 2001).

The main ideas behind the combination of a large number of de-correlated trees is that, first, the prediction error rate of a forest converges almost surely to a limit as the number of trees becomes large (Law of Large Numbers) and, second, the individual tree's prediction ability increases as correlation between the individual trees reduces. Latter strategy is enabled through randomly selecting only a subset of predictors at a split, in contrast to bagging which considers all predictors (Breiman, 2001). Random forests are, thus, particularly popular as they combine high-variance, low-bias models, such as trees, with the advantage of averaging noisy data to reduce the variance.

Due to the prediction accuracy in highly imbalanced data, random forest approaches are popularly used in the biomedical research area and in biomass mapping. Jia et al. (2016), for instance, used random forest classifiers in combination with a voting system to develop a novel ensemble classifier for predicting protein succinylation sites in protein sequences, which could potentially be used in treating cancer and other diseases. In the area of land remote sensing, Rodriguez-Galiano et al. (2012) used random forest to map complex landcover and land use at a classification accuracy of 92%. In combination with text data, random forest frameworks could also be used to effectively categorize text documents with dozens of topics more effectively than other common text categorization methods, which is demonstrated in the work of Xu et al. (2012). A recent study by Elagamy et al. (2018) found that combining random forest classification with text mining on financial news articles could lead to highly accurate indicators for predicting abnormal stock market movements.

3. METHODOLOGY

For our empirical analysis, we implemented our four machine learning algorithms, LSTM network, standard deep feedforward neural network, random forest and logistic regression, for predicting directional return movements for the stocks of S&P 500, from 2016-12-01 until 2019-12-31, and compared those models in terms of their prediction accuracy.

Our methodology comprised the following steps: first, we pre-processed the S&P 500 data to obtain complete daily returns for the entire period of 2016-12-01 until 2019-12-31. We then split the data into rolling blocks of one-calendar-month training and testing windows. Considering our time period, we got 36 calendar months as training and testing periods. Second, we defined two target variants for classifying the daily return in our training and testing sets into class 0 or 1. Third, we trained the models on the training set using several input feature sets that contained an increasing number of stock return lags (the highest lag in the feature set was 1, 3, 7, 10, 15 and 20, respectively). Fourth, on the test sets we derived the predicted target from the predicted probability of the trained models and compared them to the observed target by computing the prediction accuracy.

The final parameters we used for fitting have been selected after parameter tuning. The computations were performed in R, the programming language for statistical computing.

In the following subchapters, we describe our methodology in more detail.

3.1. DATA AND PRE-PROCESSING

Our models were trained on complete daily returns of the stock constituents of S&P 500. The current list of constituents of S&P 500 were scraped from Wikipedia using the R package **rvest**. With the function **getSymbols** from the R package **quantmod**, we downloaded the closing price from 2016-12-01 to 2019-12-31 from Yahoo Finance for all stocks that were constituents in this time frame and grouped the data by stocks and date.

The closing price of a stock s was then used for computing the daily returns for s at time t, r_t^s:

$$r_t^s = \frac{closeprice_t^s}{closeprice_{t-1}^s} - 1$$

The data was additionally pre-processed to include only stocks that are complete within the given time frame (i.e. stocks for which only partial data was available were excluded in the analysis). Our final pre-processed data frame contained 486 stocks from S&P 500.

Figure 7: *Return data after pre-processing, grouped by date and stock*

```
# A tibble: 375,390 x 3
   Date       Stock   return
   <fct>      <fct>    <dbl>
 1 2016-12-01 A     -0.0175
 2 2016-12-02 A      0.0190
 3 2016-12-05 A      0.0114
 4 2016-12-06 A      0.00696
 5 2016-12-07 A      0.00335
 6 2016-12-08 A      0.0180
 7 2016-12-09 A      0.0109
 8 2016-12-12 A     -0.00346
 9 2016-12-13 A      0.00542
10 2016-12-14 A     -0.00539
# ... with 375,380 more rows
```

Our pre-processed data from part 3.1. (see **Figure 7**) was then split into rolling blocks of one calendar month, whereby one month was used for fitting our prediction models and the month thereafter was used for testing the fitted models with new data. Thus, we split our data into two-month time intervals of one-month training and one-month testing windows (see an abstract of rolling blocks in **Figure 8**), resulting in 36 monthly training and testing windows with a different number of trading days. Our decision to split the underlying data into such short time blocks was guided by our observation that more data did not improve the model accuracy but increased running time significantly.

Figure 8: *Example illustration of one-month rolling periods of training and testing windows. In our analysis, the training period started from 1st of December 2016 and the testing period one month afterwards. We thus trained and tested 36 month-windows, whereby each month contained a different amount of trading days*

```
Month 1
Train 2016-12-01 2016-12-31
Test 2017-01-01 2017-01-31

Month 2
Train 2017-01-01 2017-01-31
Test 2017-02-01 2017-02-28

Month 3
Train 2017-02-01 2017-02-28
Test 2017-03-01 2017-03-31

Month 4
Train 2017-03-01 2017-03-31
Test 2017-04-01 2017-04-30

Month 5
Train 2017-04-01 2017-04-30
Test 2017-05-01 2017-05-31
```

3.2. MODEL TARGETS

We computed two variants of targets for our models.

Variant A:

In the first variant, the target at time t for some stock s, y_t^s, was based on the cross-sectional median return of all stocks at time t. The target was thereby defined as follows:

$$y_t^s = \begin{cases} 1 & r_t^s > m_t \\ 0 & else \end{cases},$$

whereby r_t^s is the non-standardized return of a stock s at time t and $m_t = median\{r_t^1, \dots, r_t^{486}\}$ is the cross-sectional median return (not standardized!) of all stocks (486 in total in our data) in the training resp. testing set.

Thus, if a stock return at time t was above the cross-sectional median at time t it was assigned to class 1, and 0 else. The goal was to predict whether the stock performed better than the average. Note that the return need not be positive, but the class can still be 1.

Variant A was used in the paper by Fischer and Krauss (2018).

Variant B:

In our second approach, we defined the target at time t for some stock s, y_t^s, to be class 1 simply if return at time t was positive, and 0 else. Hence, the goal was to predict whether the stock went up or down. This is also referred to as *direction-of-change forecasting*. Formally,

$$y_t^s = \begin{cases} 1 & r_t^s > 0 \\ 0 & else \end{cases}$$

This target was considered in the work of Basak et al. (2019) which analysed the prediction performance of random forests in terms of directional stock price movements. The authors found that under this target definition random forests could outperform other machine learning algorithms from literature (such as logistic regression, ANN and XGBoost).

3.3. INPUT FEATURES FOR THE PREDICTION MODELS

For fitting and testing our machine learning models, we generated input features based on an increasing number of stock return lags. The highest lag in the feature sets we considered was 1, 3, 7, 10, 15 and 20, respectively. For example, 7 input features imply that for day t

of some stock s the feature vector was defined by x_t^s = (r_{t-1}^s,, …, r_{t-7}^s) (see **Figure 9**). Note that the x_t^s on the first day(s) of a training or testing window contains returns from the previous month.

The input features were used as inputs in fitting our models in order to generate weights that minimized our loss function on the training set. The trained models were then tested on the test set to observe the prediction performance in terms of prediction accuracy.

To observe if our prediction performance was influenced by feature transformation, we ran our analysis based on standardized and non-standardized input features likewise. We performed the following standardization: each feature column was standardized separately such that per stock the feature had mean 0 and standard deviation 1. We then compared the result to non-transformed, original input features.

Figure 9: *An extract from the training set of December 2016 and the respective test set of January 2017. The target in this figure was computed based on the (non-standardized) cross-sectional median return at time* t*. The lag-return feature columns in the training resp. testing set (here, number of feature columns is* 7*) were used as input features for the prediction models*

$train
A tibble: 9,700 x 11
Groups: Date [20]

	Date	Stock	return	return.lag1	return.lag2	return.lag3	return.lag4	return.lag5	return.lag6	return.lag7	target
	<fct>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2016-12-02	A	0.0190	-0.0175	-0.0112	0.0202	-0.0287	0.00538	0.00427	-0.0113	1
2	2016-12-05	A	0.0114	0.0190	-0.0175	-0.0112	0.0202	-0.0287	0.00538	0.00427	1
3	2016-12-06	A	0.00696	0.0114	0.0190	-0.0175	-0.0112	0.0202	-0.0287	0.00538	1
4	2016-12-07	A	0.00335	0.00696	0.0114	0.0190	-0.0175	-0.0112	0.0202	-0.0287	0
5	2016-12-08	A	0.0180	0.00335	0.00696	0.0114	0.0190	-0.0175	-0.0112	0.0202	1
6	2016-12-09	A	0.0109	0.0180	0.00335	0.00696	0.0114	0.0190	-0.0175	-0.0112	1
7	2016-12-12	A	-0.00346	0.0109	0.0180	0.00335	0.00696	0.0114	0.0190	-0.0175	0
8	2016-12-13	A	0.00542	-0.00346	0.0109	0.0180	0.00335	0.00696	0.0114	0.0190	1
9	2016-12-14	A	-0.00539	0.00542	-0.00346	0.0109	0.0180	0.00335	0.00696	0.0114	1
10	2016-12-15	A	0.00910	-0.00539	0.00542	-0.00346	0.0109	0.0180	0.00335	0.00696	1

... with 9,690 more rows

$test
A tibble: 9,700 x 11
Groups: Date [20]

	Date	Stock	return	return.lag1	return.lag2	return.lag3	return.lag4	return.lag5	return.lag6	return.lag7	target
	<fct>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2017-01-03	A	0.0204	-0.00175	-0.00175	-0.0170	0.00671	0.00500	-0.00152	-0.00368	1
2	2017-01-04	A	0.0131	0.0204	-0.00175	-0.00175	-0.0170	0.00671	0.00500	-0.00152	1
3	2017-01-05	A	-0.0119	0.0131	0.0204	-0.00175	-0.00175	-0.0170	0.00671	0.00500	0
4	2017-01-06	A	0.0312	-0.0119	0.0131	0.0204	-0.00175	-0.00175	-0.0170	0.00671	1
5	2017-01-09	A	0.00313	0.0312	-0.0119	0.0131	0.0204	-0.00175	-0.00175	-0.0170	1
6	2017-01-10	A	-0.000831	0.00313	0.0312	-0.0119	0.0131	0.0204	-0.00175	-0.00175	0
7	2017-01-11	A	0.0239	-0.000831	0.00313	0.0312	-0.0119	0.0131	0.0204	-0.00175	1
8	2017-01-12	A	-0.0148	0.0239	-0.000831	0.00313	0.0312	-0.0119	0.0131	0.0204	0
9	2017-01-13	A	0.00350	-0.0148	0.0239	-0.000831	0.00313	0.0312	-0.0119	0.0131	1
10	2017-01-17	A	-0.00760	0.00350	-0.0148	0.0239	-0.000831	0.00313	0.0312	-0.0119	0

... with 9,690 more rows

3.4. MODEL EVALUATION

Based on the input features and respective target variant, our prediction models computed a target probability $\hat{f}(x_t^s)\ \epsilon\ (0,1)$, which was transformed into a target class $\hat{y}_t^s\ \epsilon\ \{0,1\}$ as follows:

Under target variant A:

$$\hat{y}_t^s = \begin{cases} 1 & \hat{f}(x_t^s) \geq\ \hat{m}_t \\ 0 & else \end{cases},$$

whereby $\hat{m}_t = median\left(\hat{f}(x_t^1), \dots, \hat{f}(x_t^{486})\right)$ is the cross-sectional median of the predicted probabilities and 486 is the total number of stocks in our data.

This means, the predicted class is 1, if the predicted probability is larger than the predicted cross-sectional median at time t.

Under target variant B:

$$\hat{y}_t^s = \begin{cases} 1 & \hat{f}(x_t^s) \geq\ 0.5 \\ 0 & else \end{cases}$$

This means, the predicted class is 1, if the model predicts that the return is positive with a probability larger than 0.5.

Prediction accuracy:

The models' prediction accuracy was computed as the average number of correct classifications for each stock in the test set. Formally,

$$prediction\ accuracy = \frac{1}{486M} \sum_{s=1}^{486} \sum_{t=1}^{M} 1_{\{y_t^s = \hat{y}_t^s\}},$$

whereby 1 denotes the indicator function, y_t^s is the observed target class for some stock s at time t, $\hat{y}_t^s$ is the predicted target class for some stock s at time t, M is the total number of trading days in the test period, and 486 is the total number of stocks in the test data.

3.5. IMPLEMENTATION IN R

In the following subchapters, we describe the implementation of our models using R. Thereby, we ran our analysis for a different number of input features ($\{1, 3, 7, 10, 15, 20\}$) with and without standardization and with target variant A and B, separately. This means, we had in total $6 \times 2 \times 2 = 24$ different settings for training.

3.5.1. LOGISTIC REGRESSION

The logistic regression for our binary example was implemented in R on the training set using the **glm()** function by setting the argument `family` to `binomial`. For each stock s separately, we computed the model by $glm(target \sim r_{t-1}^{s} + \cdots + r_{t-nlags}^{s})$ on the training set. This means, we trained in total 486 models. The **predict()** function with `type="response"` was used to predict the class probability per stock on the test data using the fitted model.

3.5.2. RANDOM FOREST

Random Forest was implemented in R from the package **xgboost** which uses gradient boosting at its core but can be adjusted through its parameter settings to run a simple random forest on the training set. For the model to run, both the training and the testing set need to be converted into a matrix by the function **xgb.DMatrix()**. For training a simple random forest the function `xgb.train` from the package **xgboost** can be used under the parameters `nrounds` = 1 and `num_parallel_tree` > 1. To grow classification trees, the parameter is set to `booster` = "`gbtree`". For our binary model, we additionally set the argument `objective` to "`binary:logistic`".

After parameter tuning, we kept the following parameter settings:

```
params = list(
  "booster" = "gbtree",
  "objective" = "binary:logistic",
  "eta" = 1,
  "max_depth" = 10,
  "num_parallel_tree" = 10,
  "subsample" = 0.5,
  "eval_metric" = "error")
```

With this parameter setting, we fitted the random forest on the training data on all stocks. This means, we only trained one model, whereby the stock information was added as another input feature. This model was used for prediction of the test set.

3.5.3. STANDARD DEEP NEURAL NETWORK

For building a DNN, we used functions from the package `tensorflow` and `keras`.

We built the model using the function `keras_model_sequential` from the `keras` package, which constructs a feedforward standard DNN. Motivated by the work of Fischer and Krauss (2018), we used 20 neurons in the first, 10 in the second, 5 in the third hidden layer, and 1 neuron (for class 1) in the output layer.

Our activation function was ReLU (rectified linear unit) in all layers except the output layer for which we used a sigmoid activation function (as we chose to have only one output class). Alternatively, one could use softmax as the output activation function for two output values. Gradients were computed on a sample of size 100 (batch size) and we used 6 iterations for training (epochs).

We fitted the deep feedforward neural network on the training data for all stocks. This means, we only trained one model. This model was used for prediction of the test set.

3.5.4. LSTM NETWORK

We computed the LSTM network model similarly to DNN using the packages `tensorflow` and `keras`.

Our trained LSTM network had the following setup:

- In the input layer the stock return lags (input features) were iteratively added to the model, in contrast to all the previous models that got the input features at once.
- The LSTM layer had $h = 10$ hidden neurons. No dropout value as weight regularization was used, otherwise the results got worse (the dropout value is used to define how many weights should be randomly selected for refresh while the others are dropped out).
- An output layer (dense layer) had 1 neuron and the sigmoid activation function.

The fitting was done similarly to DNN, with 6 number of iterations and 100 data blocks used for tuning.

As before, we fitted one model on the training data for all stocks and this model was then used for prediction of the test set.

RESULTS AND COMPARISON

Figure 10 illustrates the prediction accuracy based on target variant A and a different number of input features (number input feature was 1, 3, 7, 10, 15 and 20, respectively) with and without standardization (standardized feature is TRUE resp. FALSE). **Figure 11** shows the same settings but based on target variant B instead of A.

For both target variants, we additionally considered a purely naïve target prediction for comparison (see model NAÏVE in **Figure 10** and **Figure 11**), defined as:

$$\hat{y}_t^s = y_{t-1}^s$$

This means, the predicted class $\hat{y}_t^s$ of a stock s is equal to y_{t-1}^s, which is the observed target of the stock s on the day before. This can be viewed as a special case of logistic regression, where we set the weight of the lag 1 return to 1 and all remaining weights to zero.

In the following, we describe the findings of our simulations:

1) We observe that there is no difference in the prediction accuracy between our models, LSTM, DNN, RF and LG, as well as the naïve strategy, with respect to our time period January 2017 until December 2019.

2) The prediction accuracy of all models, including the naïve approach, is close to 0.5 (see red dotted line in the illustrations), which means that the models are as good as a purely random classifier. This implies that there is no information in the features, that are in our analysis the most recent $\{1, 3, 7, 10, 15, 20\}$ trading days prior to a predicted value.

3) For variant B we observe a strong volatility in prediction accuracy throughout the different number of input features. This could be the result of a high discrepancy between the training and test sets. For instance, if the return is generally increasing in one month, but goes down in the following month, the trained model is biased. This does not happen for target variant A, because 50% of the targets are always in class 1 and 50% are in class 0 independent of the market trend in the training set.

4) A higher number of input features has no significant influence on the prediction results. Unfortunately, not even the most recent time period considered for prediction (such

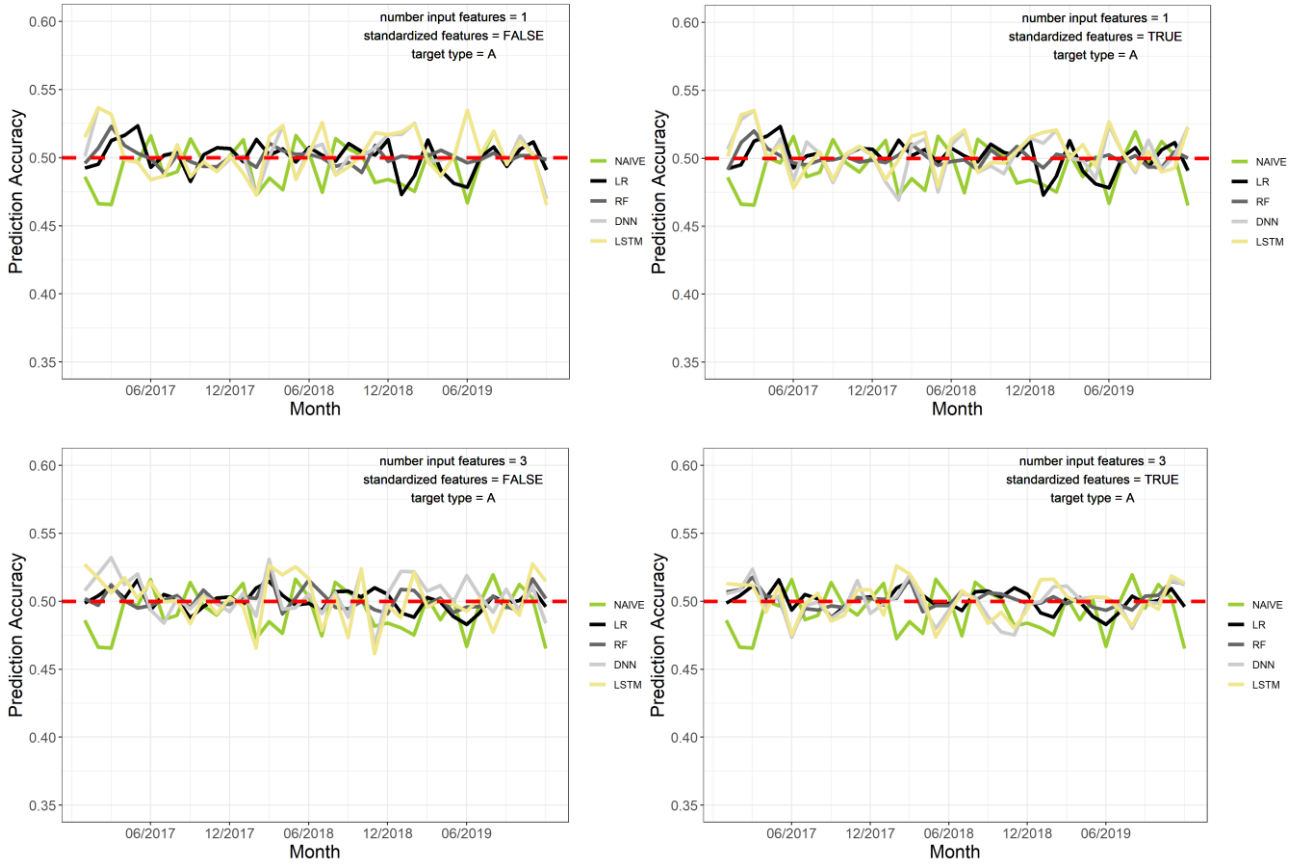
as observing yesterday to predict today) could lead to a higher prediction accuracy, which indicates that our target was entirely independent of our input features.

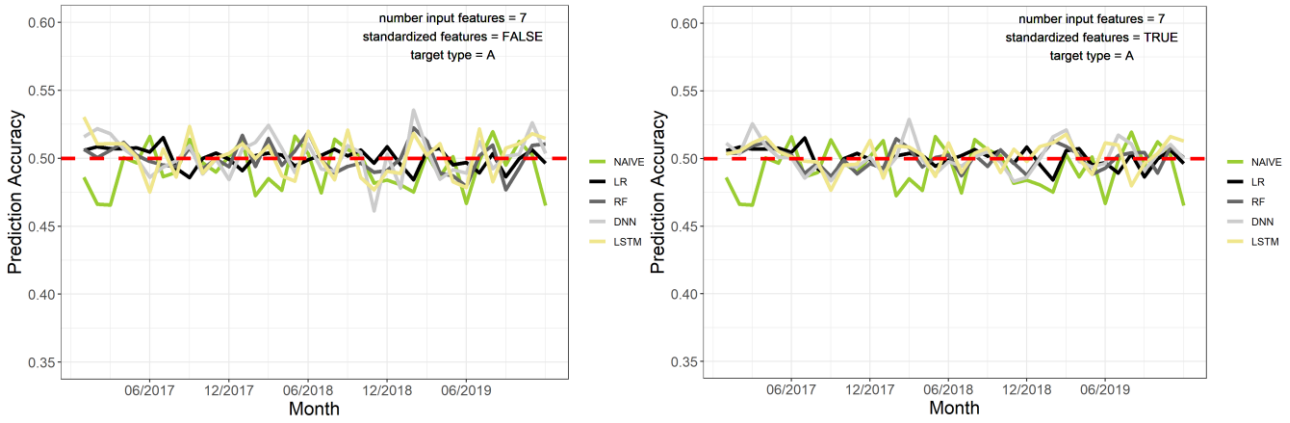
5) Our observations suggest that the longer the time frame considered, the generally higher the risk of overfitting, which can be seen in the number of times the models predicted wrongly for a higher number of input features.

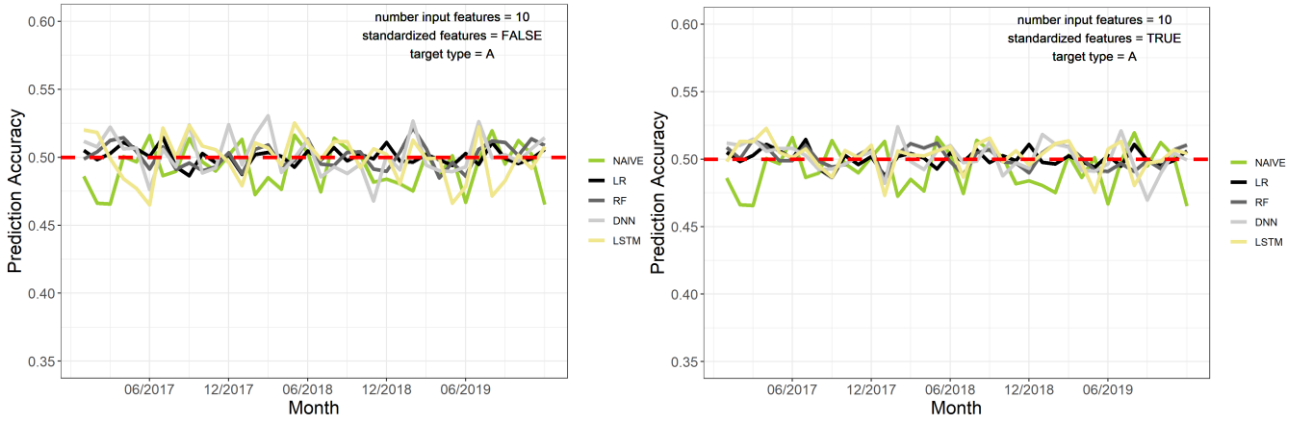
6) The variance on the prediction accuracy is slightly higher if the input features are not standardized. This suggests that standardizing features provides more robust predictions.

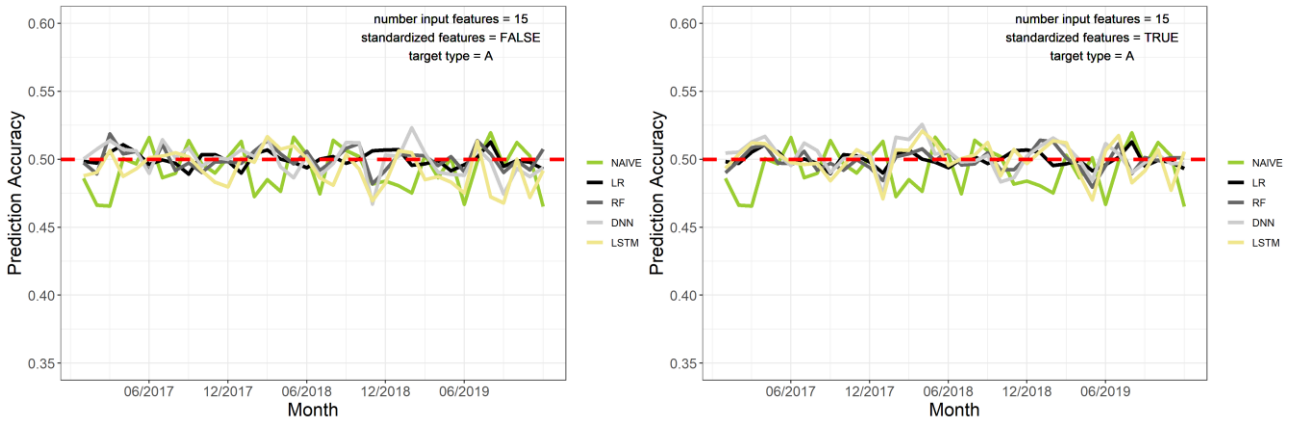
Surprisingly, not even the conservative method, logistic regression, could outperform a purely random coin flip. The more advanced machine learning models did not yield better results while showing a strong tendency to overfit to noise.

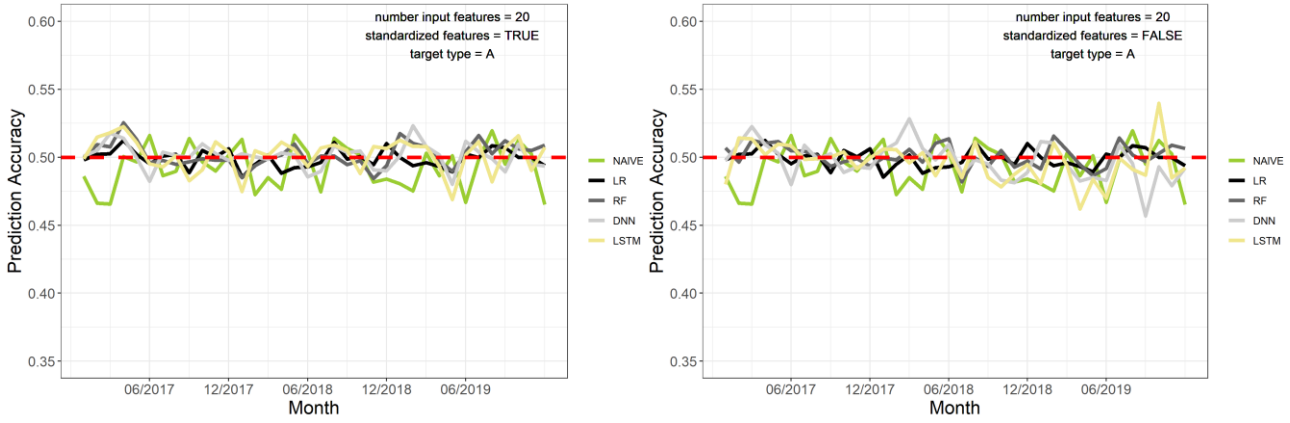
Figure 10: *Prediction results of the four models LSTM network, logistic regression (LR), random forest (RF) and standard deep neural network (DNN), from January 2017 until December 2019. The target was defined as the cross-sectional median (variant A). In the graphs to the left, the input features were not standardized; in the graphs to the right, the input features were transformed to have a mean* 0 *and standard deviation* 1




number input features = 7
standardized features = FALSE
target type = A
Prediction Accuracy
0.60
0.55
0.50
0.45
0.40
0.35
06/2017
12/2017
06/2018
12/2018
06/2019
Month
NAIVE
LR
RF
DNN
LSTM


number input features = 10
standardized features = FALSE
target type = A
Prediction Accuracy
0.60
0.55
0.50
0.45
0.40
0.35
06/2017
12/2017
06/2018
12/2018
06/2019
Month
NAIVE
LR
RF
DNN
LSTM


number input features = 15
standardized features = FALSE
target type = A
Prediction Accuracy
0.60
0.55
0.50
0.45
0.40
0.35
06/2017
12/2017
06/2018
12/2018
06/2019
Month
NAIVE
LR
RF
DNN
LSTM


number input features = 20
standardized features = TRUE
target type = A
Prediction Accuracy
0.60
0.55
0.50
0.45
0.40
0.35
06/2017
12/2017
06/2018
12/2018
06/2019
Month
NAIVE
LR
RF
DNN
LSTM

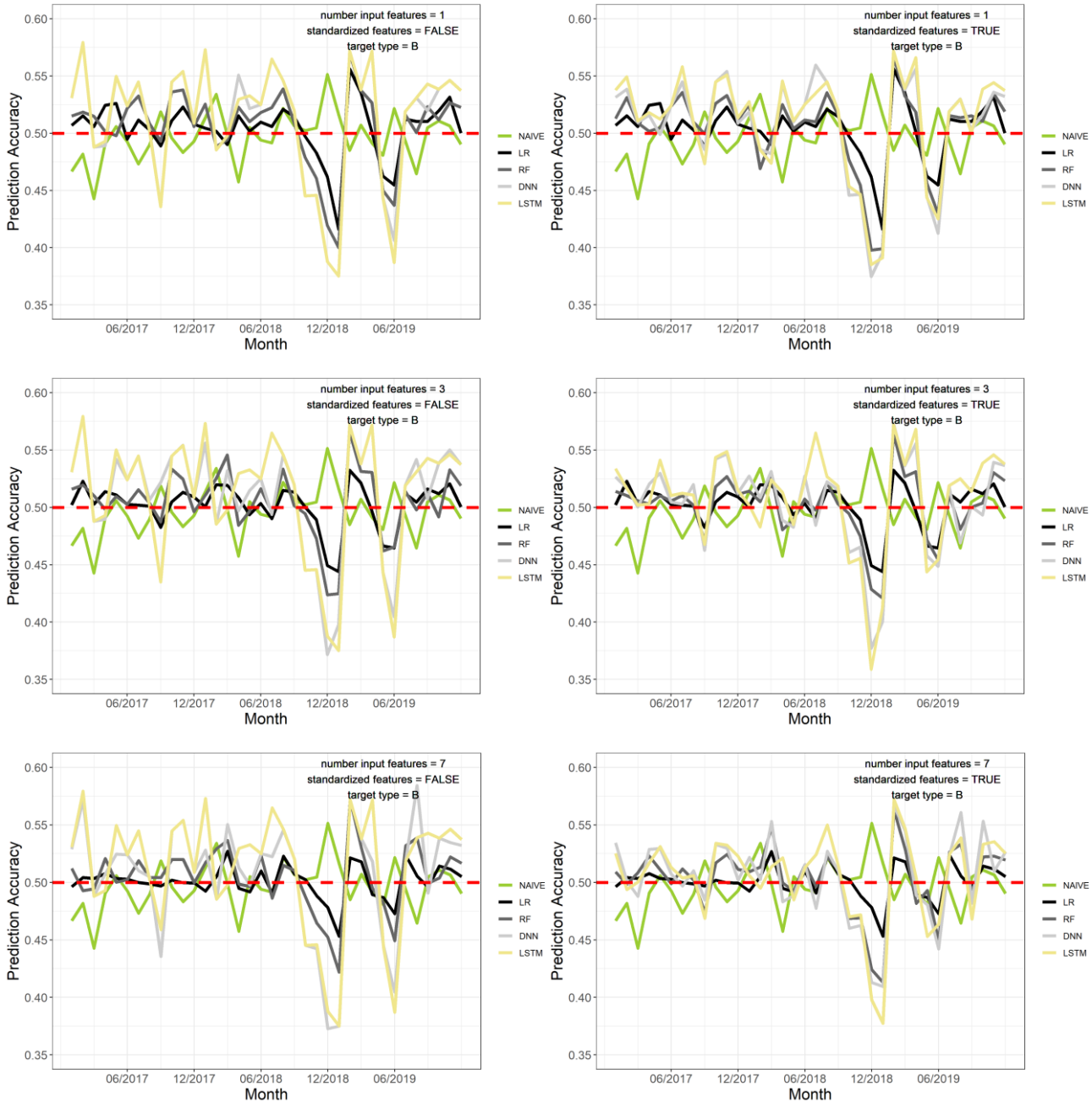
number input features = 7
standardized features = TRUE
target type = A
Prediction Accuracy
0.60
0.55
0.50
0.45
0.40
0.35
06/2017
12/2017
06/2018
12/2018
06/2019
Month
NAIVE
LR
RF
DNN
LSTM

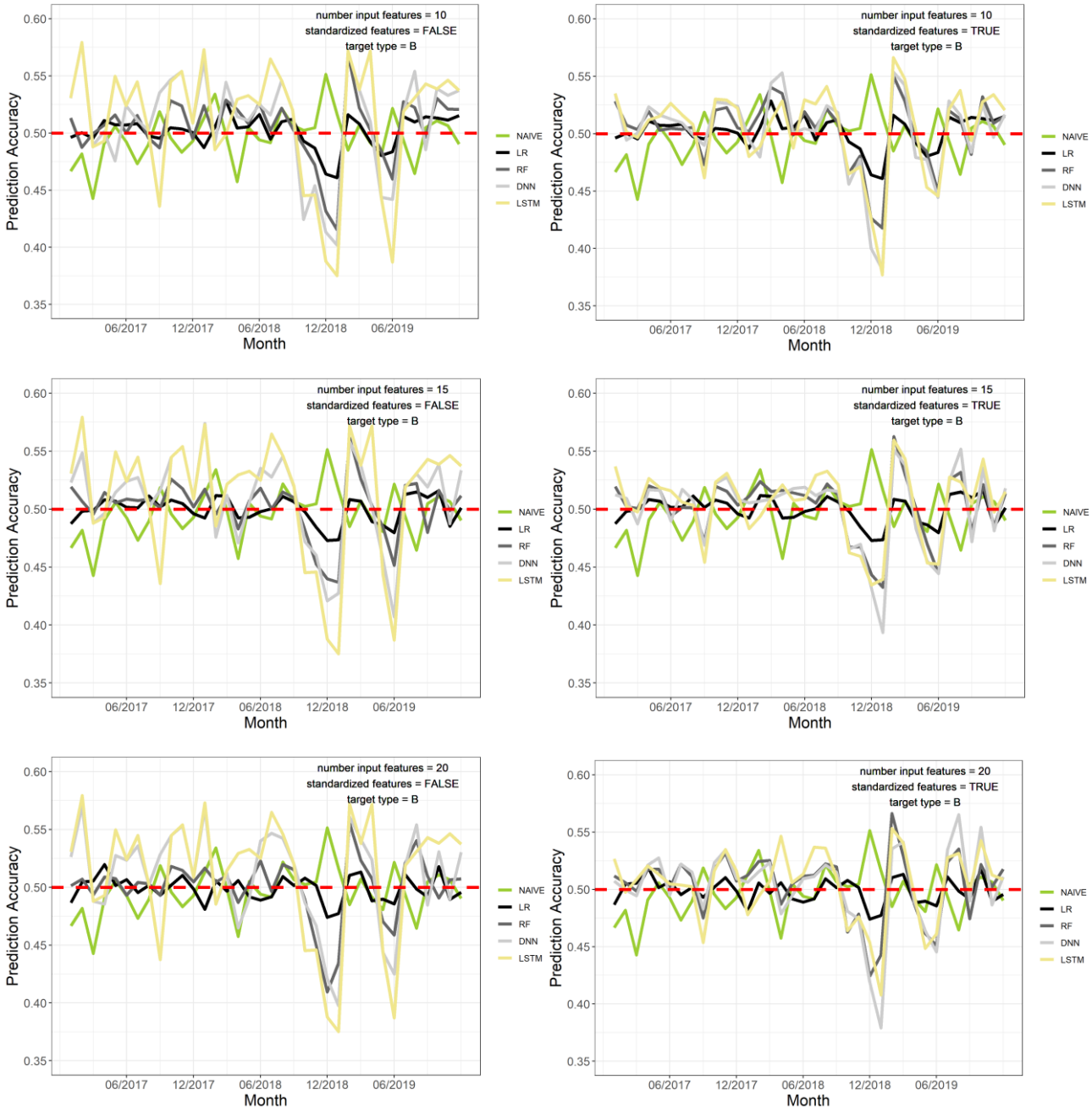
number input features = 10
standardized features = TRUE
target type = A
Prediction Accuracy
0.60
0.55
0.50
0.45
0.40
0.35
06/2017
12/2017
06/2018
12/2018
06/2019
Month
NAIVE
LR
RF
DNN
LSTM

number input features = 15
standardized features = TRUE
target type = A
Prediction Accuracy
0.60
0.55
0.50
0.45
0.40
0.35
06/2017
12/2017
06/2018
12/2018
06/2019
Month
NAIVE
LR
RF
DNN
LSTM

number input features = 20
standardized features = FALSE
target type = A
Prediction Accuracy
0.60
0.55
0.50
0.45
0.40
0.35
06/2017
12/2017
06/2018
12/2018
06/2019
Month
NAIVE
LR
RF
DNN
LSTM

Figure 11: *Prediction results under target variant B of the four models LSTM network, logistic regression (LR), random forest (RF) and standard deep neural network (DNN), from January 2017 until December 2019. The illustrations on the left show the prediction accuracy based on not-standardized features, the results in the right graphs are based on standardized features*




number input features = 10
standardized features = FALSE
target type = B
number input features = 10
standardized features = TRUE
target type = B
number input features = 15
standardized features = FALSE
target type = B
number input features = 15
standardized features = TRUE
target type = B
number input features = 20
standardized features = FALSE
target type = B
number input features = 20
standardized features = TRUE
target type = B
Prediction Accuracy
0.60
0.55
0.50
0.45
0.40
0.35
06/2017
12/2017
06/2018
12/2018
06/2019
Month
NAIVE
LR
RF
DNN
LSTM

CONCLUSION

Supervised machine learning algorithms are booming in the fields of academic and practical research. For a fast variety of applications ranging from visual and textual to numeric problem settings, they have shown to yield accurate prediction results as they improve automatically through experience and thus enable to use the most out of large and extensive data sets.

In this paper, we applied a memory-based LSTM network, a standard deep feedforward neural net, a random forest as another model benchmark as well as the conservative logistic regression as a baseline to a financial market prediction task on S&P 500, from December 2016 until December 2019. We compared the results of the four models in terms of their prediction accuracy taking into consideration pre-processing variants in feature selection and target generation. Our analysis was motivated by the contributions from Fischer and Krauss (2018) in which LSTM networks outperform memory-free classification methods in terms of predicting directional stock price movements.

In our empirical application, we found that neither conservative nor state-of-the-art machine learning models could properly predict the financial market price movements considering the volatile and uncertain stock market development of the past years. The poor prediction performance of our models was largely induced by the model's likelihood to overfit on very noisy data.

Several recent studies indicate, however, that including sentiment analysis or a combination of numerical and textual information in feature selection or correlation analysis could enhance the prediction accuracy in financial market prediction tasks. Their observations suggest that sentiments expressed through text streams in news or social media can give some insights on stock price trends (Ren et al., 2018; Elagamy et al., 2018; Chan and Chong, 2017; Akita et al., 2016; Pagolu et al., 2016).

We conclude that because of the instability of the stock movements in the recent years, machine learning based trading strategies are highly inaccurate in terms of directional stock movement prediction. Our results indicate that either return lags are white noise or machine learning algorithms are not capable of detecting the pattern behind the stock return features. We suggest that further research on financial market prediction in combination with sentiment analysis could possibly yield more promising results.

REFERENCES

[1] Akita R., Yoshihara A., Matsubara T. and Uehara K. (2016): Deep learning for stock prediction using numerical and textual information, published in 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), pp. 1-6

[2] Basak S., Kar S., Saha S., Khaidem L. and Dey S.R. (2019): Predicting the direction of stock market prices using tree-based classifiers, published in The North American Journal of Economics and Finance, Vol. 47, pp.552-567

[3] Breiman L. (2001): Random Forests, scientific paper, pp.1-33, retrieved from https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf, accessed on 23rd March 2020

[4] Chan S.W.K. and Chong M.W.C. (2017): Sentiment analysis in financial texts, published in Decision Support Systems, Vol. 94, pp. 53-64

[5] Chan W., Jaitly N., Le Q. and Vinyals O. (2016): Listen, attend and spell: A neural network for large vocabulary conversational speech recognition, published in The 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp.4960-4964

[6] Chong E., Han C. and Park F.C. (2017): Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies, published in Expert Systems with Applications, Vol. 83, pp. 187-205

[7] Ciresan D., Meier U., Masci J. and Schmidhuber J. (2012): Multi-column deep neural network for traffic sign classification, published in Neural Networks, Vol. 32, pp. 333-338

[8] De Mulder W., Bethard S. and Moens M.F. (2015): A survey on the application of recurrent neural networks to statistical language modelling, published in Computer Speech and Language, 30th edition, pp.61-98

[9] Deng L., Hinton G. and Kingsbury B. (2013): New types of deep neural network learning for speech recognition and related applications: an overview, published in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 8599-8603

[10] Elagamy M. N., Stanier C. and Sharp B. (2018): Text Mining Approach to Analyse Stock Market Movement, Advances in Intelligent Systems and Computing, No. 723, published in The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2018), pp. 661-670

[11] Esteva A., Kuprel B., Novoa R. A., Ko J., Swetter S. M., Blau H. M. and Thrun S. (2017): Dermatologist-level classification of skin cancer with deep neural networks, published in nature, research journal, pp. 115-118

[12] Fischer T. and Krauss C. (2018): Deep learning with long short-term memory networks for financial market predictions, published in European Journal of Operational Research 270, pp.654-669

[13] Goodfellow I., Bengio Y. and Courville A. (2016): Deep Learning, published in MIT Press, retrieved from http://www.deeplearningbook.org, accessed on 19th January 2020

[14] Graves A. (2012): Supervised Sequence Labelling with Recurrent Neural Networks, Studies in Computational Intelligence, Springer

[15] Greff K., Srivastava R. K., Koutnik J., Steunebrink B. R. and Schmidhuber J. (2017): LSTM: A Search Space Odyssey, published in IEEE Transactions on Neural Networks and Learning Systems, Vol. 28, No. 10, pp. 2222-2232

[16] Gu J., Wang Z., Kuen J., Ma L., Shahroudy A., Shuai B., Liu T., Wang X., Wang G., Cai J. and Chen T. (2018): Recent advances in convolutional neural networks, published in Pattern Recognition, Vol. 77, pp. 354-377

[17] Hastie T., Tibshirani R. and Friedman J. (2008): The Elements of Statistical Learning: Data Mining, Interference, and Prediction, 2nd edition, published in Springer Series in Statistics, retrieved from https://web.stanford.edu/~hastie/Papers/ESLII.pdf, accessed on 19th January 2020

[18] Hochreiter S. (1991): Untersuchungen zu dynamischen neuronalen Netzen, diploma thesis in Computer Science at the Institut for Computer Science at the Technical University of Munich, pp.1-215

[19] Hochreiter S. and Schmidhuber J. (1997): Long short-term memory, published in Neural Computation, Vol. 9, No. 8, pp. 1735-1780

[20] Howard A. G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., Andreetto M. and Adam H. (2017): MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, published in https://arxiv.org/pdf/1704.04861.pdf, pp.1-9

[21] Jia J., Liu Z., Xiao X., Liu B. and Chou K.C. (2016): pSuc-Lys: Predict lysine succinylation sites in proteins with PseAAC and ensemble random forest approach, scientific paper, published in Journal of Theoretical Biology, Vol. 394, pp. 223-230

[22] Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q. and Liu T. (2017): LightGBM: A Highly Efficient Gradient Boosting Decision Tree, published in Advances in Neural Information Processing Systems 30 (NIPS 2017)

[23] Keras (2020): Layer activation functions, retrieved from https://keras.io/api/layers/activations/, accessed on 30th May 2020

[24] Long W., Lu Z. and Cui L. (2019): Deep learning-based feature engineering for stock price movement prediction, published in Knowledge-Based Systems, Vol. 164, pp. 163-173

[25] Nelson D.M.Q., Pereira A.C.M. and De Oliveira R.A. (2017): Stock market's price movement prediction with LSTM neural networks, published in 2017 International Joint Conference on Neural Networks (IJCNN), pp. 1419-1426

[26] Nielsen A.M. (2015): Neural Networks and Deep Learning, published in Determination Press, retrieved from http://neuralnetworksanddeeplearning.com/chap5.html, accessed on 19th January 2020

[27] Pagolu V. S., Reddy K. N. Panda G. and Majhi B. (2016): Sentiment analysis of Twitter data for predicting stock market movements, published in 2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), pp. 1345-1350

[28] Ren R., Wu D.D. and Liu T. (2018): Forecasting Stock Market Movement Direction Using Sentiment Analysis and Support Vector Machine, published in The IEEE Systems Journal, Vol. 13, No. 1, pp. 760-770

[29] Rodriguez-Galiano V.F., Ghimire B., Rogan J., Chica-Olmo M. and Rigol-Sanchez J.P. (2012): An assessment of the effectiveness of a random forest classifier for

land-cover classification, scientific paper, published in ISPRS Journal of Photogrammetry and Remote Sensing, Vol. 67, pp. 93-104

[30] Salehinejad H., Sankar S., Barfett J., Colak E. and Valaee S. (2017): Recent Advances in Recurrent Neural Networks, published in https://arxiv.org/pdf/1801.01078.pdf, pp.1-21

[31] Sze V., Chen Y.H., Yang T.J. and Emer J.S. (2017): Efficient processing of deep neural networks: A tutorial and survey, published in The Proceedings of the IEEE, Vol. 105, Issue 12, pp. 2295-2329

[32] Xie S., Girshick R., Dollar P., Tu Z. and He K. (2017): Aggregated Residual Transformations for Deep Neural Networks, published in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1492-1500

[33] Xu B., Guo X., Ye Y. and Cheng J. (2012): An Improved Random Forest Classifier for Text Categorization, published in Journal of Computers, Vol. 7, No. 12, pp.2913-2920

ABSTRACT ENGLISH

This paper examines the prediction performance of machine learning models on stock price movements of S&P 500 stocks, from 2017 until 2019, in a binary classification problem. We apply four machine learning algorithms for predicting daily return based target classes: a standard deep feedforward neural network, a long short-term memory (LSTM) network as a special case of a recurrent neural network, random forest and logistic regression. We additionally consider a naïve prediction strategy for comparison. As input of the machine learning algorithms, we consider daily stock return lags. As model target, we consider two variants: one target variant is defined as class 1 if the stock return is positive, and 0 else (also known as direction-of-change). The other target variant is defined as class 1 if the stock return is above the cross-sectional median of all stocks, and 0 else. We have several simulation settings: with and without standardization of input features and with an increasing number of stock return lags. Our experimental results show that in all of the settings neither the machine learning algorithms nor the naïve strategy outperform a purely random classification of a coin flip in terms of prediction accuracy. We conclude that using machine learning algorithms for predicting price movements based on historic price data alone is insufficient and contains a high risk of overfitting.

ABSTRACT GERMAN

Diese Arbeit vergleicht die Prognosegenauigkeit von Machine Learning Algorithmen auf Preisbewegungen von Aktien aus S&P 500 im Zeitraum 2017 bis 2019. Im Rahmen eines binären Klassifizierungsproblems setzen wir vier Machine Learning Modelle zur Prognostizierung von Klassen ein, die auf den Tagesreturns basieren: ein Standard Deep Neural Network, ein Long Short-Term Memory (LSTM) Neural Network als Sonderfall von rekurrenten neuronalen Netzen, sowie Random Forest und Logistische Regression. Wir betrachten zusätzlich eine naïve Prognosestrategie als Benchmark. Als Input unserer Modelle verwenden wir zeitverzögerte Aktienreturns. Als Zielvariable im Modell ziehen wir zwei verschiedene Varianten für die Klassifizierung heran: in der einen Variante ist die Zielvariable als Klasse 1 definiert, wenn der Aktienreturn positiv ist und 0 sonst (auch bekannt als direction-of-change). In der anderen Variante ist die Zielvariable als Klasse 1 definiert, wenn der Aktienreturn über dem Querschnittsmedian aller Aktien liegt und 0 sonst. Mehrere Simulationsparameter werden betrachtet: wir verwenden zum einen Inputvariablen mit und ohne Standardisierung, zum anderen eine unterschiedliche Anzahl an zeitversetzten Aktienreturns als Inputvariablen. Unsere Ergebnisse zeigen, dass in allen Simulationen weder die Machine Learning Algorithmen, noch die naïve Methode besser sind als ein Münzwurf in Hinblick auf die Prognosegenauigkeit. Wir schließen daraus, dass Machine Learning Modelle auf Basis von historischen Aktienkursen zur Prognostizierung von Preisbewegungen unzureichend sind und ein hohes Overfitting-Risiko bergen.