Application and Comparison of different Machine-Learning Methods for Financial Time Series Forecasting at the Examples of the DAX30 and the S&P500

Thesis · October 2015

DOI: 10.13140/RG.2.2.21542.52807

1 author:

Deniz Ersan
Christian-Albrechts-Universität zu Kiel
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Application and Comparison of different Machine-Learning Methods for Financial Time Series Forecasting at the Examples of the DAX 30 and the S&P 500

Master's Thesis

for the Master's degree programme Quantitative Finance
in the Faculty of Business, Economics and Social Sciences at the
Christian-Albrechts-University Kiel

submitted by
Deniz Ersan

First Assessor: Prof. Dr. Ansgar Scherp, Knowledge Discovery, Department of Computer Science

Second Assessor: Prof. Dr. Thomas Lux, Chair of Monetary Economics and International Finance, Department of Economics

Kiel, October, 2015

Contents

List of Figures

List of Tables

List of Abbreviations

ANN Artificial Neural Network

DAX30 Deutscher Aktienindex (German Stock Index)

DJIA Dow Jones Industrial Average

DS Directional Symmetry

EMA Exponential Moving Average

EUR Euro

GMT Greenwich Mean Time

GPU Graphical Processing Unit

ICA Independent Component Analysis

k**-NN** k-Nearest Neighbor

KPCA Kernel Principle Component Analysis

MA Moving Average

MLP Multilayer Perceptron

NMSE Normalized Mean Square Error

NN Neural Network

OOS Out-of-sample

PCA Principle Component Analysis

RMSE Root Mean Square Error

ROC Rate of Change

RPROP Resilent Propagation

S&P 500 Standard & Poor's 500 Index

SVC Support Vector Classification

SVM Support Vector Machine

SVR Support Vector Regression

USD US-Dollar

1 Introduction

Forecasting financial time series is regarded as one of the most challenging assignments in the domain of modern time series forecasting [1]. According to the prominent Efficient Market Hypothesis, market prices (in efficient markets) reflect all available information of a stock at any time. Prices only adapt on the arrival of new information. Hence, forecasting future stock price movements based on historical information should not enable to generate excess returns [2]. Despite the prominence of the Efficient Market Hypothesis, numerous researches question its validity [3, 4, 5].

Also, Timmermann and Granger (2004) [5] consider a possible existence of an "file drawer" bias in published studies, because publishing empirical results that are barely or just statistically insignificant may be out of interest for researchers. Meanwhile, the continuous advance of machine learning techniques and computational power gave birth to machine learning approaches applied to financial forecasting problems. Encouraging results have been obtained by artificial neural networks (ANNs) [6], support vector machines (SVMs) [1] and k-nearest neighbor method (k-NN) [7]. Krollner et al. (2010) [8] provide an survey of recent literature of machine learning approaches applied to financial time series forecasting. However, existing studies usually do not use data with higher than daily sampling rates. Further, only few studies apply further techniques, for example dimensionality reduction or ensemble methods, to improve the predicting power of the learning machines.

In this thesis, we evaluate and compare the performance of various machine learning approaches for forecasting daily and hourly index level returns of the German stock index (DAX 30) and the Standard & Poor's 500 (S&P 500) stock index. Furthermore, we analyze optimal training window lengths for the three methods, which may reveal notion about the learning behavior of the three learning machines. Further, we apply a method for dimensionality reduction to the training data, namely the kernel principle component analysis (KPCA), and investigate its impact on the learning algorithms. Additionally, we examine the influence of the bootstrap aggregating algorithm, which ensembles multiple trained machines in order to improve stability and accuracy. Finally, we use selected learning machines for building a simple trading system solely based on the predictions generated by the learning algorithms.

This thesis contains six sections and is organized as follows. In Section 2, we introduce the basic concepts of machine learning and our selected algorithms, namely SVMs, ANNs, k-NN, KPCA and Bootstrap Aggregating, also referred to as "bagging". Furthermore, we provide an overview of related work on financial time series forecasting and machine learning. Section 3 explains our experimental approach for comparing our algorithms and highlights research questions. The results of the experiment are outlined in Section 4. In Section 5 we interpret our empirical results in order to draw inferences regarding our research questions. Finally, Section 6 presents conclusions, limitations and an outlook for future research.

2 Theoretical Background

2.1 Machine Learning

Machine learning belongs to the field of artificial intelligence and it combines ideas from a broad variety of disciplines, such as neuroscience and biology, statistics, mathematics and physics [9]. Machine learning can be defined as the creation of computer-based systems, which are capable of learning from experience in order to solve predetermined tasks evaluated by some performance measure, if its performance enhances with experience [10, 11].

For instance, we could design a machine, the task of which is to predict if the stock market will be up or down at the end of the next trading day based on preceding stock index movements. The performance could be measured by a simple hit ratio which reports how often the machine was right or wrong on predicting the next days direction.

In this simple example learning would take place, as we provide historical stock index data to let the machine gather experience for improving the accuracy of its predictions. Learning refers to remembering, adapting and generalizing from experience or more specifically data [9], which is typically conducted by algorithms. There is a considerable number of machine learning algorithms that differ in the way they learn and the type of output they can produce [9, 10]. Therefore, we group various algorithms according to their characteristics in the learning process.

2.1.1 Supervised Learning

The learning process is referred as supervised learning if the learning process involves a teacher who has a knowledge of the environment, where knowledge can be represented by paired input-output examples. This knowledge is yet unknown to the machine. Given the inputs, the machine is provided the outputs by the teacher, which represent the true responses the machine is expected to produce [12].

The input-output examples, also called training data, are of the form (x_{ti}, y_i), where x_{ti} denotes the input variables and y_t the output variable. The index $t, t = 1, ..., T$ declares the iteration of the example, i.e. a time stamp and the index $i, i = 1, ..., I$ indicate the number of the input variable. Within every iteration during the training, the machine seeks to produce a response as close as possible to the true responses. On this way the knowledge of the environment, which was previously only available for the teacher is transferred to the machine. This procedure is called error correction learning [12].

Another objective of this learning process is to tune the free parameters for the learning algorithm, which again involves the teacher. Clearly, there is an inevitable high degree of human supervision required during the learning process, but accompanied by sufficiently large training sets, supervised learning algorithms can achieve noteworthy results [12]. A rather novel field of machine learning algorithms, called semi-supervised learning addresses the issue of (costly) human supervision. The aim of semi-supervised learning is to reduce human supervision by combining labeled data and unlabeled data [13]. A comprehensive introduction to semi-supervised learning can be found in [13].

Supervised learning algorithms can be applied for classification and regression purposes [9]. In Section 2.2 we explain the learning algorithms which, we apply in our experiment in Section 4.

2.1.2 Unsupervised Learning

In contrast to supervised learning, which is always guided by a teacher, unsupervised learning does not involve a teacher in the learning process [12]. Unsupervised learning algorithms work with unlabeled data. More specifically, given the inputs, again of the form x_{ti} with $t, t = 1, ..., T$ and $i, i = 1, ..., I$ the machine is not taught any correct responses. Instead, the learning algorithm attempts to detect similarities among the inputs in order to group inputs with common characteristics [9].

At this stage, the question arises how unsupervised learning algorithms actually learn. More distinctly, what performance measure improves with the experience of the machine. The answer is that unsupervised learning machines are able to produce representations of raw input data such that redundancy is reduced, which is essential for perceiving massive flows of information intelligently [14].

A typical goal in unsupervised learning is dimensionality reduction, while analyzing data or preprocessing data for later use [9]. In Section 2.2.4 we provide a short introduction to (kernel) principle component analysis, which is a popular approach in unsupervised learning.

2.1.3 Reinforcement Learning

We learned that unsupervised learning machines do not involve a teacher. Reinforcement learning does not come with a teacher either. Instead, the learning problem is performed by an agent, who is faced with decisions under uncertainty in a set environment, which gives responses in form of rewards to the agent [15]. Thus, reinforcement learning corresponds to learning with a critic. The goal is to learn an input-output mapping through the interaction of the agent and the environment that minimizes a predetermined performance measure [12].

Figure 1 depicts the interaction process of the agent within its environment. At every discrete time step t the agent is given the current state $s_t, s_t \in S$ via the environment from the set of all possible states S. Now, the agent has to decide which action $a_t, \in A(s_t)$ given the set of all possible actions, given the current state s_t it executes. Once the agent executed action a_t the environment responds to that

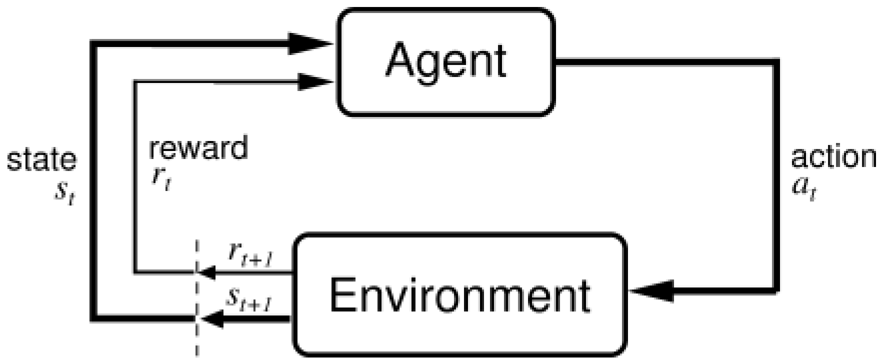


Figure 1: The agent-environment interaction in reinforcement learning. Source: [15]

action with a reward r_{t+1} and returns the next state s_{t+1}. This interaction routine repeats itself over and over, while the agent seeks to maximize the total sum of the rewards.

Reinforcement learning methods are out of scope of this thesis and will not be applied in our experiment. For a more detailed introduction to reinforced methods, see [12, 15].

2.1.4 Evolutionary Learning

Evolutionary learning algorithms introduce principles of evolutionary theory such as the survival of the fittest mechanism to machine learning. They are also referred to as stochastic search algorithms [16]. There are two characteristics which distinguish evolutionary algorithms from other search algorithms. First, they do not only consider a single solution point as in conventional optimization. Instead, they consider a set of possible solution points. Second, individuals in a population communicate and exchange information [17].

Possible solutions to the optimization problem are referred as individuals in a population, and the goal is to improve the solutions through evolution of the population [17]. The performance of a candidate solution is measured by a predefined fitness function. Evolutionary algorithms differ regarding what sort of evolutionary mechanism they use [17]. Commonly used evolutionary algorithms are the genetic algorithm, the evolution strategy and the swarm optimization algorithm.

Further introduction to the field of evolutionary learning is left to the reader as evolutionary learning will not be considered within the scope of this thesis. A comprehensive introduction to the topic of evolutionary learning and genetic algorithms can be found in [18, 19].

2.2 Selected Algorithms

In Section 2.1 we provided an introduction to machine learning including a classification of the variant types of learning algorithms. Furthermore, we emphasized how learning takes place when learning comes with a teacher (supervised learning) and without a teacher (unsupervised learning). The aim of this Section is to give the reader a sound intuition of the learning algorithms, which we apply later on in our empirical experiments in Section 4.

2.2.1 Support Vector Machine

A SVM is a supervised learning technique for data analysis and pattern recognition. The history of SVMs goes back to 1963 when Vapnik and Lerner [20] introduced a linear classifier that separates training samples with the widest margin producing an optimal separating hyperplane. In 1993, Guyon et. al [21] extended the optimal hyperplane by allowing for nonlinear separation by utilizing kernels. Conclusively, Cortes and Vapnik [22] proposed a method, which can deal even with non-separable data. SVMs can be applied to both, classification and regression problems. In the following we provide an introduction to both concepts. For a detailed introduction, we recommend [22, 23, 24]. Our notation is close to [22, 23].

Support Vector Classification (SVC)

Linear Separable Case The goal of classification[1] is to train a machine, given labeled observations, so it can produce out-of-sample predictions of y, as soon as new

[1]Here: Binary classification without loss of generality for extension

observations of the input variables are fed to the machine. Therefore, we need to find a hyperplane that separates the observations according to their class membership. The labeled data are of the following form:

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_l, y_l)\}, x \in \mathbb{R}^n, y \in \{-1, +1\} \tag{2.1}$$

The vector $\mathbf{x_i}$ represents a l-dimensional input vector. The inputs are labeled with the associated true outputs y_i, which can take only two values, for instance $y \in \{+1; -1\}$. Further, we assume that the training observations are randomly and independently distributed, from an unknown function. In the separable case we require all observations to satisfy:

$$\mathbf{x_i} \cdot \mathbf{w} + b \geq +1 \quad \text{for } y_i = +1 \tag{2.2a}$$

$$\mathbf{x_i} \cdot \mathbf{w} + b \leq -1 \quad \text{for } y_i = -1, \tag{2.2b}$$

where $\mathbf{w}$ is a perpendicular vector, $\cdot$ refers to the dot product and b to some bias. Both constraints (2.2a) and (2.2b) simply ensure that the data is separable. Combining both we can write:

$$y_i(\mathbf{x_i} \cdot \mathbf{w} + b) - 1 \geq 0 \ \ \forall i \tag{2.3}$$

Suppose, we have a two-dimensional input space. Clearly, there is an infinite number of possible hyperplanes separating the observations. A SVM seeks to find the optimal hyperplane, which is the one with the maximum margin while separating the observations [23]. In Figure 2 we can see the optimal hyperplane and another random hyperplane. Both satisfy (2.3). Obviously, the optimal hyperplane will give a better classification performance for new observations, provided that the new observations follow the same unknown function as the training observations, since the optimal hyperplane leads to the maximum separation between both classes [25]. The points shaping any upper (lower) hyperplane $H1$ ($H2$) lead to:

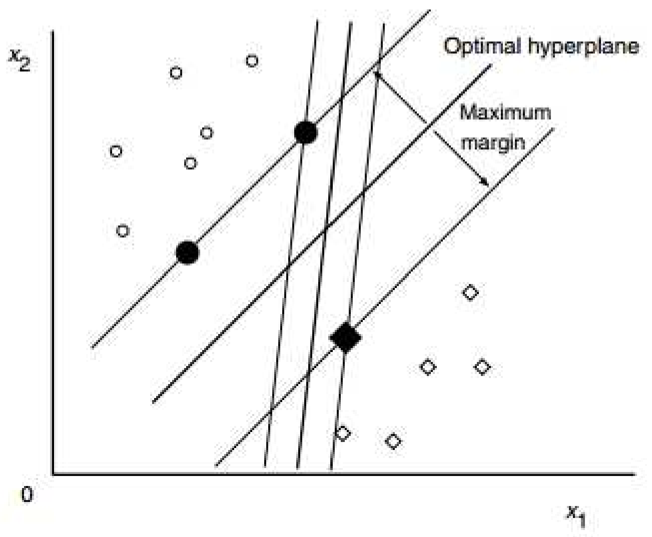


Figure 2: Optimal separating hyperplane in a two-dimensional space for linear separable data. The bold symbols are the support vectors. Source: [25]

.

$$H1 : \mathbf{x_i} \cdot \mathbf{w} + b = 1 \tag{2.4a}$$

$$H2 : \mathbf{x_i} \cdot \mathbf{w} + b = -1 \tag{2.4b}$$

Then, we can find the optimal hyperplane by maximizing the sum of the upper and the lower margins $\frac{1}{\|\mathbf{w}\|} + \frac{1}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$ with respect to (2.3). Minimizing the norm of $\mathbf{w}$, denoted by $\|\mathbf{w}\|$, involves a square root which can be replaced by $\frac{1}{2}\|\mathbf{w}\|^2$ [25], leading to a quadratic objection function with linear constraints. This quadratic optimization problem can be defined as (2.5) [26]:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{subject to: } y_i(\mathbf{x_i} \cdot \mathbf{w} + b) - 1 \geq 0 \ \ \forall i \tag{2.5}$$

At this stage, it is convenient to introduce lagrange multipliers $\boldsymbol{\alpha}$ to (2.5), since a lagrangian formulation of the problem is easier to solve and essential for for the nonlinear extension of the SVM [23]. The primal form of the constrained problem with in the lagrangian representation is:

$$L_P = \frac{1}{2}\left\|\mathbf{w}\right\|^2 - \sum_{i=1}^{l} \alpha_i \left[y_i(\mathbf{x_i} \cdot \mathbf{w} - b) - 1 \right] \tag{2.6}$$

(2.6) has to be minimized with respect to $\mathbf{w}$ and b, and maximized with respect to $\boldsymbol{\alpha} \geq 0$. After setting,

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i}^{l} \alpha_i y_i x_i \tag{2.7a}$$

and

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow b = \sum_{i}^{l} \alpha_i y_i = 0, \tag{2.7b}$$

we can formulate the Lagrangian in the dual form via substitution.

$$L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \underbrace{\mathbf{x_i} \cdot \mathbf{x_j}}_{\text{training data}}, \text{subject to} \begin{cases} \alpha_i \geq 0 \ \forall i \\ \alpha_i y_i = 0 \end{cases} \tag{2.8}$$

In contrast to (2.6), we need to maximize (2.8) as their constraints are different. Since we set $b = 0$ in (2.7b), all optimal hyperplanes will contain the origin [23]. Although b only depends on x_i and y_i, it can be computed by taking the mean from all i with nonnegative α_i. The solution will give $\mathbf{w} = \sum_i^l \alpha_i y_i x_i$ in both forms and $b = \frac{1}{N_{SV}} \sum_i^{N_{SV}} \mathbf{x_i} \cdot \mathbf{w} - y_i$, N_{SV}: Number of support vectors. We highlighted that the training data only appear in form of dot products in (2.8) as it will be of use later. The linear classifier for the separable case can be defined as:

$$f(\mathbf{x}) = sgn(\mathbf{x} \cdot \mathbf{w}) + b, \tag{2.9}$$

where sgn is the sign function that extracts the sign of the solution from the bracketed dot product.

Linear Non-Separable Case A strong shortcoming so far is that our SVM is not able to give a solution for non-separable data. In order to handle non-separable data we need to amend the constraints (2.2a) and (2.2b) by introducing nonnegative slack variables $\xi_i, i = 1, ..., l$ [22, 23].

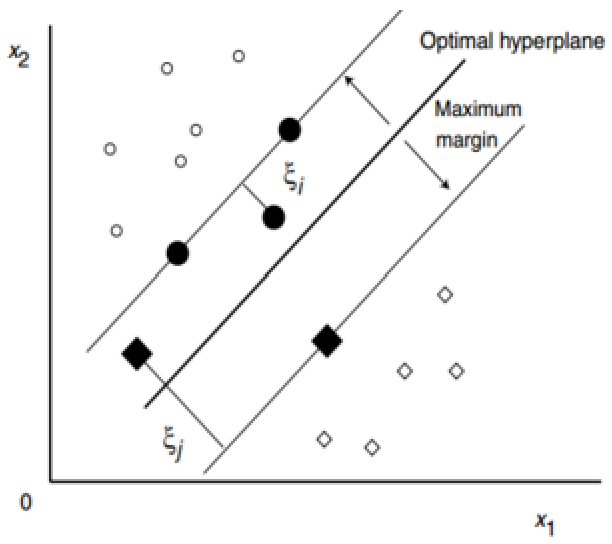


Figure 3: Optimal separating hyperplane in a two-dimensional space for non-separable data. Source: [25]

We introduce nonnegative slack variables, so our constraints for non-separable data change to:

$$\mathbf{x_i} \cdot \mathbf{w} + b \geq +1 - \xi_i \quad \text{for } y_i = +1 \tag{2.10a}$$

$$\mathbf{x_i} \cdot \mathbf{w} + b \leq -1 + \xi_i \quad \text{for } y_i = -1 \tag{2.10b}$$

$$\xi_i \geq 0 \quad \forall i \tag{2.10c}$$

If an error in classification occurs, the corresponding ξ_i will exceed 1. In Figure 3 we can see the difference between two observations lying inside the separating hyperplane. Note that ξ_i is not greater than 1 and hence it is not misclassified as ξ_j which exceeds 1. Still, ξ_i has an impact as it decreases the maximal margin [25]. We add a penalty parameter to our previous quadratic optimization problem in order to assess errors while optimizing.

$$\arg\min_{\mathbf{w},b}\left\{\frac{1}{2}\left\|\mathbf{w}\right\|^2 + C\sum_{i=1}^{l}\xi_i\right\}, \text{subject to}\begin{cases} y_i(\mathbf{x_i}\cdot\mathbf{w}+b)-1\geq 1-\xi_i \quad \forall i, \\ \xi_i\geq 0 \end{cases} \tag{2.11}$$

The free parameter C can be set individually by the supervisor, depending on how much penalty is supposed to be assigned to errors. However, the decision has to be made wisely as C can affect the generalization performance significantly and is subject to a proper optimization method of the SVM. The corresponding dual form of the Lagrangian representation and its solution are identical to (2.8). Only one of the constraints changes. We have:

$$L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \underbrace{\mathbf{x_i}\cdot\mathbf{x_j}}_{\text{training data}}, \text{subject to}\begin{cases} 0\geq\alpha_i\geq C \ \forall i \\ \sum_i \alpha_i y_i = 0 \end{cases} \tag{2.12}$$

The solution is again $\mathbf{w} = \sum_i^l \alpha_i y_i x_i$ and for b we have $b = \frac{1}{N_{SV}}\sum_i^{N_{SV}} \mathbf{x_i}\cdot\mathbf{w} - y_i$. Also, the classifier remains

$$f(\mathbf{x}) = sgn(\mathbf{x}\cdot\mathbf{w}) + b$$

for the non-separable case.

Nonlinear Non-Separable Case Up to now, we only considered linear classifiers, that would perform poorly when applied to data with nonlinear patterns. Guyon et al. (1993) [21] proposed the nonlinear extension to SVMs by utilizing the kernel method, originally invented by Aizermann et al. (1964) [27]. Due to their work, the transition from linear to nonlinear SVM is quite simple.

Previously, we highlighted that the training data only appear in form of dot products in our algorithm. Therefore, the idea is to replace the dot products $\mathbf{x_i}\cdot\mathbf{x_j}$ in (2.8) by some nonlinear kernel function which maps the training data to some higher dimensional Euclidean space. While the input vector is mapped in a higher dimensional space $\boldsymbol{x} \mapsto \Phi(\mathbf{x})$, we now search for linear separation in this higher dimensional space. According to Cover's theorem [28] we are more likely to find linear patterns while mapping our data to a higher dimensional space than in a lower dimension.

So we need to replace all dot products $\mathbf{x_i} \cdot \mathbf{x_j}$ by some kernel $K(\mathbf{x_i}, \mathbf{x_j}) = \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$. That will give us the nonlinear classifier.

$$f(\mathbf{x}) = \sum_{i=1}^{N_{SV}} \alpha_i y_i K(sv_i, \mathbf{x}) + b \tag{2.13}$$

Support Vector Regression (SVR) Previously we looked at classification problems where the output y was either $+1$ or -1. By contrast, regression estimation deals with estimating real-valued functions allowing the output $y \in \mathbb{R}$ to take any real number. Usually, square error functions are applied to regression problems, which enable least-squares estimation. In 1995, Vapnik [22] introduced a proper method, that addresses the loss of accuracy from large residuals caused by outliers and allows for noise in data. The corresponding loss function (2.15) is an extension of Huber's loss function (2.14) [25].

$$L(y, \hat{y}) = |y - \hat{y}| \tag{2.14}$$

$$L_\epsilon(y, \hat{y}) = \begin{cases} |y - \hat{y}| - \epsilon & \text{for } |y - \hat{y}| \geq \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

The ϵ is a free parameter and can be set individually by the supervisor, just as the parameter C. The term $|y - \hat{y}|$ is the absolute distance between the desired output y and the predicted output $\hat{y}$. In Figure 4 we can see that only points outside the grey shaded area add a penalty to the optimization process. Of course, we can apply (2.15) for nonlinear decision surfaces as well.

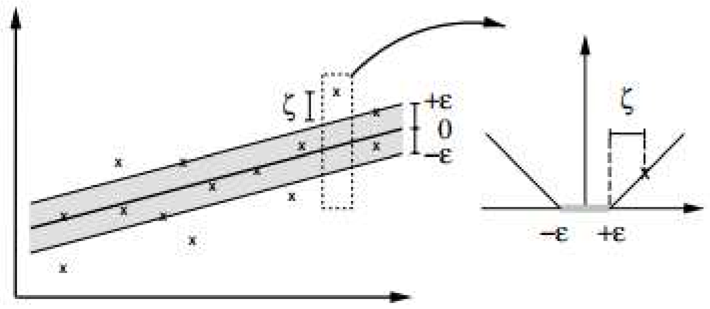


Figure 4: ϵ-insensitive loss function for a linear SVM. Source: [29]

2.2.2 Artificial Neural Network

An ANN is a machine learning method inspired by the functioning of the human brain. Basically the aim of an ANN is to learn a nonlinear mapping $\mathcal{F}_{NN} : X \mapsto Y$, producing a real-valued, discrete-valued or vector-valued function, where X represents a d-dimensional input matrix and Y denotes the output variable [11, 30]. A more detailed introduction to ANNs can be found in [11, 30].

Artificial Neuron In biology a neuron is a cell that is able to transmit and process information. The human brain consists of approximately 10^{11} neurons, while each neuron is connected to 10^4 other neurons [11]. An artificial neuron consists of an input, a cell body and an output which is illustrated in Figure 5. Each input x_i has a weight w_i which represents the strength of that particular connection. The sum of weighted inputs including the bias b is transmitted to the activation function f to produce the output y. This can be written as:

$$y = f\left(\sum_{i=1}^{n} x_i w_i + b\right) \tag{2.16}$$

Activation Function The activation function f defines the amplitude of the output y of a neuron, given the inputs x_i and weights w_i. The most widely used activation functions are sigmoid functions. Depending on their definition they can be

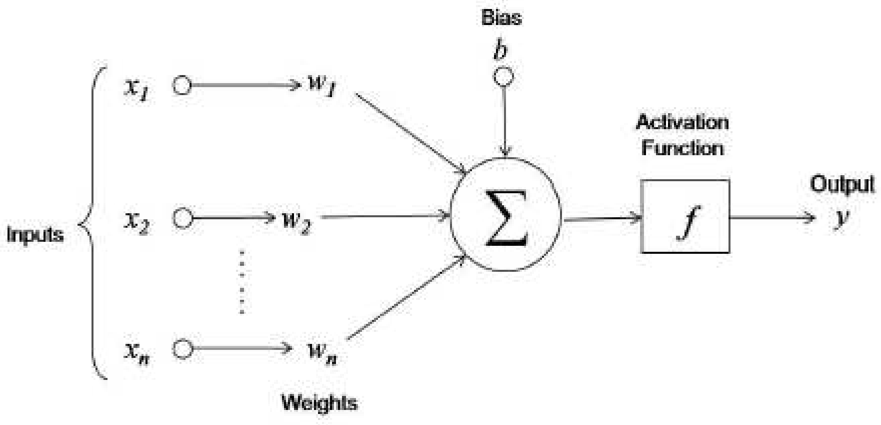


Figure 5: An Artificial Neuron

referred to as log sigmoid function, defined for $(0, 1)$, or hyperbolic tangent sigmoid function for $(-1, +1)$. A linear activation function is also applicable, which is defined for all real numbers.

$$f(x) = \frac{1}{1 + e^{-x}} \qquad \text{Log Sigmoid Function} \tag{2.17a}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad \text{Hyperbolic Tangent Sigmoid} \tag{2.17b}$$

Network Topology Neural Networks can be classified in feedforward NN and recurrent NN. In feedforward NN data only flow in one direction, more specifically from input nodes to the output, while recurrent NN involve cycles or loops. In the following we focus on feedforward NN, as it is the network topology applied in Section 4. A multilayer feedforward NN consists of an input layer, an output layer and at least one hidden layer between the two as illustrated in Figure 6. The neurons in the layer are connected to the neurons in the following layer, thus each neuron receives the input from the preceding layer and transmits the output to the next layer. Within the same layer there is no connection between the neurons.

Learning and Generalization As mentioned before the objective of an ANN is to learn a mapping of inputs and outputs. Learning happens while training the machine, by iteratively adjusting the weights of ANN according to the error between the actual and the desired output, until the minimum error solution is found. There are several suitable algorithms for solving the mapping, but they differ in speed and accuracy. The most promising algorithm is the resilient propagation (RPROP) algorithm, which is widely used, since it outperforms many local adaptive algorithms in speed and accuracy [31]. The RPROP uses first order optimization on each weight independently, but it only considers the sign, not the amplitude of the partial derivatives of the error function with respect to the weight. Since initial weights have to randomized and the

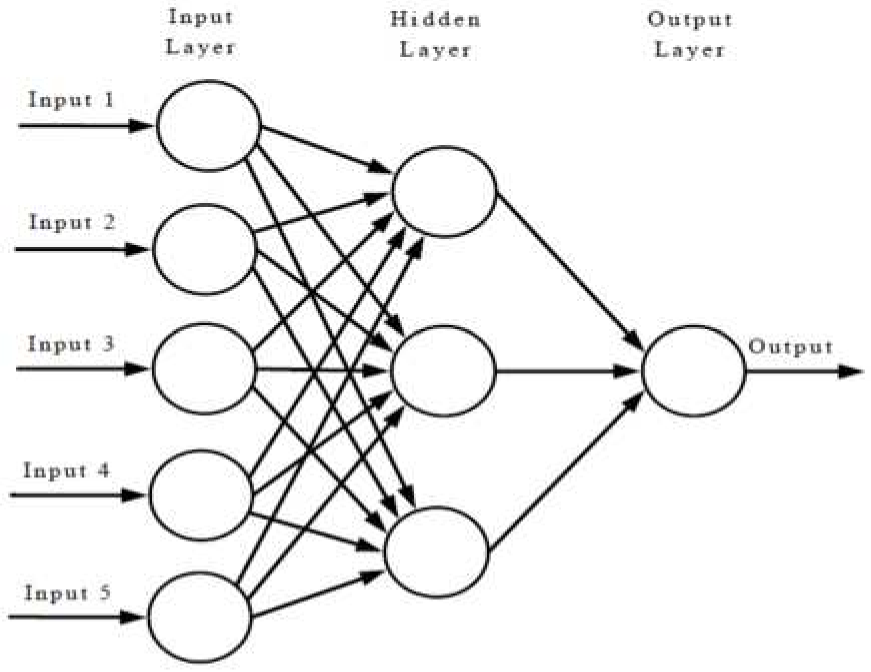


Figure 6: Multilayer Feedforward Neural Network

2.2.3 k-Nearest Neighbor

Both, SVMs and ANNs produce an explicit function once they are trained. In contrast, the k-NN method simply stores training examples and does not generalize from the training data until a new observation, or a new instance must be classified. This principle is called Instance-based learning. When the k-NN algorithm is encountered with a classification or regression query it starts to work out training examples, which are close to the new observation [9].

In order to assess the nearest neighbors, the k-NN algorithm has to compute all Euclidean distances from the new observation to each training example to determine the k closest training examples. The Euclidean distance is defined as:

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2} \tag{2.18}$$

where x denotes an arbitrary instance which is described by the feature vector $\langle a_1(x), a_2(x), ..., a_n(x) \rangle$, and $a_r(x)$ denotes the value of the r-th feature of x. After computing the distances, the learning machine searches for the k nearest neighbors. The output can be either a class membership for classification tasks or a real valued number for regression tasks.

For classification the k-NN algorithm allows the k-nearest neighbors to vote for the class membership of the new observation, while in regression it determines the real value output by taking the averages of the k-nearest neighbors [11]. As one may imagine, the choice of k is crucial for the machine. A large k is more capable to deal with noise in the data, but a small k will give a smoother boundary [32]. For choosing k one can apply the well known bootstrap technique [33].

2.2.4 Kernel Principle Component Analysis

The aim of KPCA is to transform the original input data, such that the dimension of the data is reduced to a smaller set, that still contains most of the information [34]. In contrast to the previous mentioned algorithms, KPCA involves unlabeled data.

Let us consider a data matrix $\mathbf{X}$ consisting of n rows and p columns. The rows contain various features and the columns contain all sample observations. Now, we standardize the matrix $\mathbf{X}$ such that each column mean is 0 and each column variance is 1. Next, we choose the direction with the largest variation of our transformed matrix, say $\mathbf{R}$ and place an axis in this direction. After this, we will look at the remaining variation and place an orthogonal axis in direction of the largest variation. We repeat this procedure until there are no more possible axes. In Figure 7 we can see how the aforementioned process works [9].

More formally, we seek a rotation matrix $\mathbf{P}$ to be multiplied with $\mathbf{R}$, that will give a new matrix $\mathbf{Y}$ with a diagonal covariance matrix.

$$\mathrm{Cov}(\mathbf{Y}) = \mathrm{Cov}(\mathbf{P}^T\mathbf{R}) = \mathbf{P}^T\mathrm{Cov}(\mathbf{R})\mathbf{P} \tag{2.19}$$

Now, we want to compute the eigenvectors $\mathbf{V}$ and eigenvalues $\mathbf{D}$ of $\mathbf{Y}$. Then we sort the matrix $\mathbf{D}$ containing the eigenvalues of $\mathbf{Y}$ in decreasing order and apply the same order to $\mathbf{D}$. Now, we are able to exclude features with eigenvalues smaller than some individually chosen threshold η, so we discard features which contain too little information [9].

The transition to nonlinear directions of variation may be useful for many applications. In order to allow for nonlinearity, we recall the kernel trick which we have used for the nonlinear SVM earlier. Again, we need to map our original data $\mathbf{X}$ to some higher dimension space via some nonlinear kernel function $\Phi(\mathbf{X})$. In this feature space, we are able to perform linear PCA. Finally, when KPCA in applied to the data, it reduces the nonlinear dimensionalitiy in the data.

2.2.5 Bootstrap Aggregating

Bootstrap aggregating, often also referred to as "bagging", belongs to the class of ensemble methods. It is the most prominent independent method of ensemble methods [35]. As the name suggests, bootstrap aggregating consists of bootstrapping and aggregating [36]. Bootstrapping is, resampling a training set T containing n paired observations of input and output combinations, into subsamples with t observations, where $t \leq n$. This can be done by randomly drawing n observations with replacement

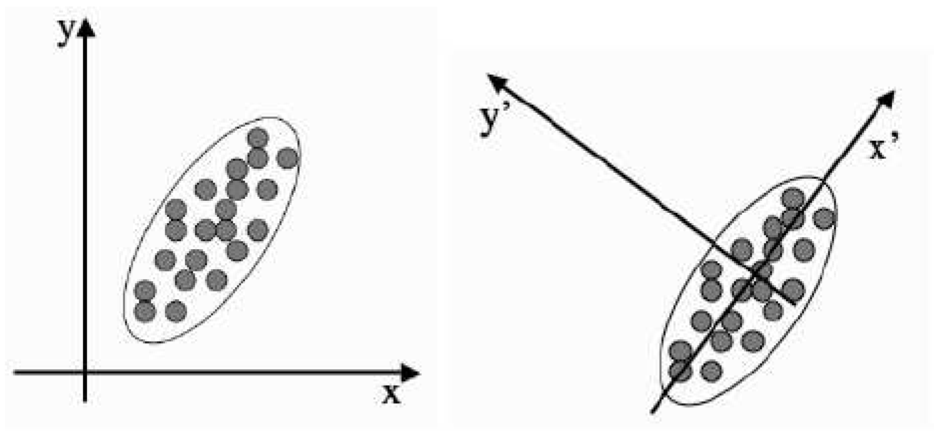


Figure 7: Left panel: Original data. Right Panel: Data with transformed axes. Source: [9]

from T and repeating this procedure for a predetermined number of times.

After bootstrapping, the idea is to use the subsamples for training supervised learning algorithms for multiple times and then use the trained machines to let them predict the output variables, given new input data. Next, aggregation can be performed. Via aggregation the predictions coming from multiple trained machines will be combined. Of course, as one could imagine, there are numerous ways to aggregate their predictions. A simple and unbiased way of aggregating is to calculate the mean over all predictions for a particular time step, so the predictions of all machines are equally weighted. In the end, the predictions are supposed to be less erroneous, since we average over multiple single predicting machines.

2.3 Bias-Variance Trade-Off

Clearly, it impossible to forecast the unknown future perfectly. We believe that the same applies to forecasting financial returns. Every model will produce errors. These errors must not be judged equally, meaning that there are smaller and larger errors. Therefore, it is of great interest to understand the sources of error for the forecasting problem at hand.

In supervised learning regressions, the mean square error (MSE) is a very prominent candidate for quantifying errors. The expectation of the MSE of a learning machine on a given test data $\mathbf{x}$ can always be decomposed into the sum of the variance of the predictor $Var(\hat{f}(\mathbf{x}))$, the squared bias of the predictor $\left[bias\hat{f}(\mathbf{x})\right]^2$ and the variance of an irreducible error $Var(\epsilon)$ [37]. For the expectation of the MSE we write:

$$E\left(y - \hat{f}(\mathbf{x})\right)^2 = Var(\hat{f}(\mathbf{x})) + \left[bias\hat{f}(\mathbf{x})\right]^2 + Var(\epsilon)\ , \tag{2.20}$$

where y refers to the corresponding true output. In order to minimize (2.20), we need to minimize the variance of the predictor and squared bias of the predictor simultaneously. These two quantities change when model flexibility is modified. In scenarios, where $f(\mathbf{x})$ is known, it can be seen that the squared bias increases, when model flexibility is decreased. In contrast, the variance will decrease [37]. Otherwise described, the supervisor of the learning machine is trading squared bias for variance when modifying the flexibility. However, calculating the test MSE explicitly is impossible in real world problems, since the true $f(\mathbf{x})$ is not observable.

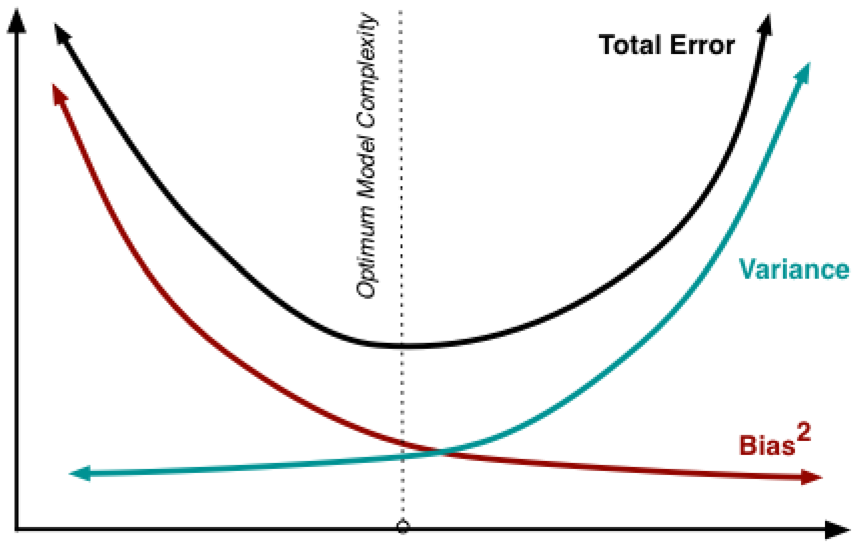


Figure 8: The bias-variance trade-off: The horizontal x-axis denotes the model flexibility and the vertical y-axis measuring the error. Source: [37]

The trade-off between square bias and variance for different levels of model complexity is illustrated in Figure 8. The total error, here the explicit test MSE, decreases in the first place, when the flexibility increases, but the error will incline after the optimal trade-off between bias and variance is reached. Finally, it is important to note that a good learning machine, at least in terms of MSE, always contains the aforementioned quantities and ideally, these should both be small. In the meantime, one should keep in mind that changing the model flexibility may not always induce a proportional trade of square bias and variance, as the absolute slopes of the two quantities are not equal for every point on the x-axis.

2.4 Related Work on Financial Time Series Forecasting

Previously, we elaborated the principles of the learning algorithms which we apply and evaluate in our experiment in Section 4. Now, it remains to review relevant studies in the domain of machine learning in a financial time series forecasting context. To that end, a plenitude of related work accumulated during the last 15-20 years. However, these studies usually differ regarding the applied learning technique, the dataset, the forecasting time-frame, the input variables and the evaluation method [8].

There is a broad variety of promising applications of ANNs for forecasting financial time series. Among these applications, the accuracy of ANNs varies significantly due to different input variables, data preparation methods and network architecture [38]. Pacelli et al. (2011) [6] utilized a genetic algorithm to find the optimal architecture for a multilayer perceptron (MLP) ANN. They used the MLP-ANN for forecasting daily EUR/USD price movements based on fundamentals and historical price data with promising results. Guresen et al. (2011) [39] evaluated the predictive power of a MLP-ANN, a dynamic artificial neural network and a hybrid neural network for forecasting daily closing levels of NASDAQ. The MLP-ANN achieved the best results in terms of mean square error and mean absolute deviation.

Cao and Tay (2001) [1] evaluated the performance of SVMs for forecasting five real future contracts compared to a MLP-ANN. They used lagged historical price returns as well as a technical indicator as input variables. The SVM was able to outperform the ANN measured by the normalized mean square error (NMSE). In [40] they can validate the dominance of the SVM over the MLP-ANN, while additionally comparing it to a radial basis function NN, which produced comparable results. Additionally, they show that a SVM with adaptive parameters can both achieve better generalization performance and use fewer support vectors than the standard SVM. The results of Cao and Tay (2001) are in line with Kim (2003)[41] who compared a SVM, a NN, and the k-NN algorithm for predicting the daily directional change of the Korean stock index.

Teixeira and De Oliveira (2010) [7] proposed an automatic stock trading system combining technical analysis and nearest neighbor classification based on daily stock prices and volumes. Their model was able to outperform a buy-and-hold strategy for the most companies measured by profitability. Furthermore, Brasileiro et al. (2013)[42] also elaborated a trading system where buy and sell signals are produced by the k-NN algorithm. Their system was able to outperform thirteen out of fifteen stocks compared to a buy-and-hold and another strategy.

In [43] Cao et al. (2003) apply PCA, KPCA, and Independent Component Analysis (ICA) to a SVM for feature extraction. They examined the sunspot data, the Santa Fe dataset A and five real future contracts and found out that their SVM can perform better with than without feature extraction, while the best performance was accomplished by KPCA followed by ICA. However, they did not apply the feature extraction method to machine learning approaches other than the SVM. Furthermore, performance was only measured by NMSE.

In the recent years, a growing number of text-mining approaches have been applied to public available text-data, like news articles, tweets on twitter or social media in general. For instance, Bollen et al. (2011) [44] investigated whether the sentiment of large-scale Twitter feeds are correlated to the value of the Dow Jones Industrial Average (DJIA) over time and they explored that daily variations in sentiment in the twitter data have a statistically significant correlation to daily Dow Jones Industrial Average close price movements [44]. However, correlations may improve predictions, but they do not represent a causation between the two. Furthermore, their data could increase prediction accuracy of a self organizing fuzzy neural network model.

Rao et al. (2013) combined twitter data and search volume index data collated from Google trends with results in line with Bollen et al. (2011) [45]. We do not not use text-data in our later experiment as it is out of scope. However, the combination of news or sentiment paired with technical indicators as input variables in training predictive machine learning models has to be further analyzed.

Not all of the aforementioned works provide results in form of trading performance of their models. In light of the prominent Efficient Market Hypothesis and for the readers interest, it is still interesting to know if the trained machines are able to generate excess returns. Also, we could not find a comparison of the SVM, the NN and k-NN approaches applied to hourly stock data. Consequently the datasets in previous works are relatively small. Not many previous experiments utilized a walk-forward training routine, where the machine is retrained after a predetermined number of OOS predictions is done.

3 Design of the Experiment

3.1 Data

The dataset used in our experiment covers the period from 02/01/2004 08:00 GMT - 06/03/2015 20:00 GMT at a hourly sampling rate, which can be converted to any lower frequency. We use hourly and daily sampling rates. It contains open-high-low-close index levels of the German stock index (DAX 30) and the American Standard & Poor's 500 index (S&P 500). The data was provided by courtesy of OANDA Corporation. In Figure 9 and 10 we can see the index-level evolution of both datasets for the whole length of our experiment. We observe that over long periods both indexes show similar index level development. Both indexes reveal up and down trends, while the up trends tend to be more consistent in terms of their length.

The output variable for our learning algorithms is the logarithmic rate of change based on the close level. It is common to transform absolute price changes into relative price changes in order to deal with more symmetric distributed data which will

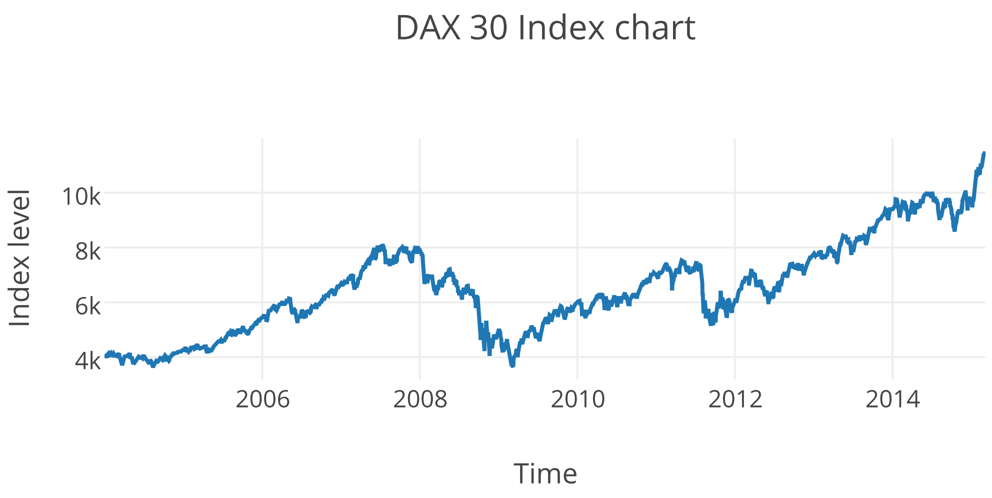


Figure 9: DAX 30 index-level chart

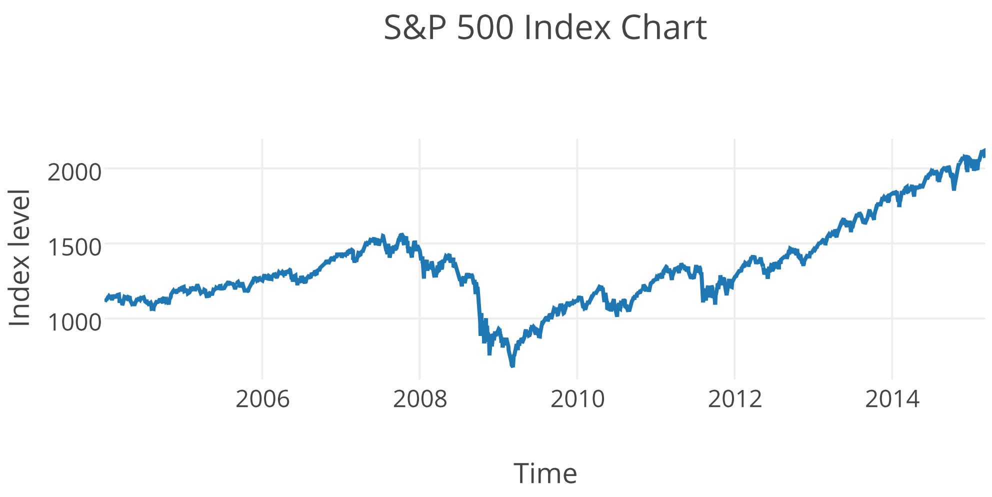


Figure 10: S&P 500 index-level chart

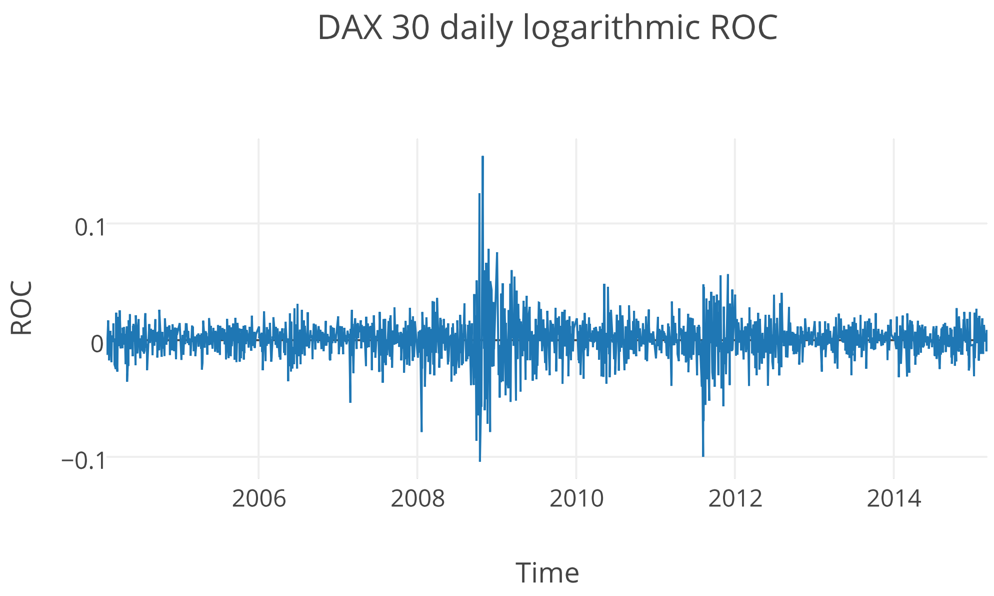


Figure 11: Output variable for the DAX 30 daily dataset

follow a normal distribution more closely [46]. By using logarithmic returns instead of arithmetic returns, we may increase performance in training, since multiplicative relationships turn into additive, which are also simpler to handle for later calculations [47].

$$r_t = ln(p_{t+1}) - ln(p_t) \tag{3.1}$$

Equation (3.1) shows the logarithmic price change with p_t denoting the close index-level in time t. The results from transforming index-level data to logarithmic ROC can be found in Figure 11, which contains the output variable for the DAX 30 daily dataset.

In Table 1 we provide some basic descriptive information for our transformed data. Please note that hourly data, due to hours of low trading activity contain hours with zero returns regularly which have been removed from the datasets. The reason is

	(a)		(b)	
	DAX	S&P	DAX	S&P
Number of observations	37160	61085	2836	3384
Mean	2.84e-05	1.01e-05	0.00037	0.00018
Std. dev.	0.00380	0.0026	0.0146	0.0113
Skewness	-0.1243	0.6780	0.0744	-0.2072
Kurtosis	17.04	50.98	11.89	14.46
Minimum	-0.0551	-0.0479	-0.1041	0.1051
Maximum	0.0472	0.0668	0.1578	0.1121
AD-Test $p-Value$	2.2e-16	2.2e-16	2.2e-16	2.2e-16
KS-Test $p-Value$	2.2e-16	2.2e-16	2.2e-16	2.2e-16

Table 1: Descriptive statistics for the return series (a) hourly and (b) daily

that, the vast majority of the data are already located very closely around zero, see Figure 11, so we rather want the machine to be trained for output observations with a non-zero value.

The difference in the number of observations for the DAX and the S&P Data is a consequence of different trading hours and days since both cover the same time span. All time series have close to zero means paired with tails, which are heavier compared to the normal distribution, quantified by the kurtosis measure being greater than 3. Our tests for normality, namely the Anderson-Darling-Test and the Kolmogorov-Smirnov-Test clearly neglect normality with very low $p-Values$. These are prominent features of financial time series, which are well documented in literature.

3.2 Feature Selection

The choice of input variables plays a major role in forecasting and its impact on the prediction performance is great. In this thesis, we aim at comparing algorithms, so our purpose is not to find a combination of input variables, that maximizes the prediction accuracy. Therefore, we decided to select similar input variables to those of previous research.

The input variables for our models are lagged technical indicators, which can be

Feature name	**Parameter**	**Selected lags**	**Calculation**
Log ROC	-	1,2,3,4,5	$ln(p_{t+1}) - ln(p_t)$
EMA ROC	$n = 3$	1,2,3,4,5	$\frac{EMA_{t+1}}{EMA_t}$
n-Day ROC		2,3,4,5	$\frac{p_{t+1}}{p_{t-n}} - 1$
MA difference	$n = 20$	1,2,3,4,5	$p_{t+1} - MA_{t+1}$

Table 2: List of our input variables and their respective calculations

extracted from our output time series. All indicators are at least lagged for one time-step, as we want to avoid look-ahead bias. For our experiment, we can only use data that would have been observable or known during training and testing in order to obtain the true forecasting capability. Table 2 gives an overview of the calculations of our selected input variables and the selected lags, as well as the selected values for the necessary parameters in some of the inputs.

We selected lagged logarithmic rates of change (log ROC) as inputs in order to allow the machine to search for patterns in the history of the output variable, which is the log ROC. In addition, we added lagged exponential moving average rates of change (EMA ROC) in order to obtain smoothed ROCs. Our third input variable is the n-day ROC, which computes the rate of change for a specified length n. For the last category of inputs, we computed differences between the current price and the current simple moving average (MA difference), again for multiple lags. We expect the MA difference to capture some of potential inter-temporal index-level trends, as it will take larger (smaller) values when the current index-level departs upwards (downwards) from the current 20-day moving average index-level.

In total, we have 19 input variables for training our machines to predict the log ROC of the next time step.

3.3 Data Preprocessing and Walk-Forward Routine

An important step before implementing and comparing our machine learning algorithms is to create a systematic approach for data preprocessing, training and prediction. We decided to use a walk-forward routine for all of our algorithms. In our walk-forward routine, we divide the whole dataset into smaller and equally sized subsets, which are used to train and test our algorithms. A subsample contains $n+i$ observations, with n denoting the length of the training window and i the number of desired out-of-sample (OOS) predictions. Once the respective algorithm is trained on the training window with length n it will produce i OOS predictions. Afterwards, the training window moves i time steps forward and the algorithms is trained with the new training window and generates new predictions.

Basically, the data for training our algorithms is simply "walking forward", while adding new observations and dropping old observations. The walk-forward routine continues to move forward until the dataset is used entirely, more specifically, until there are not enough observations to produce the disered number of OOS predictions. Since the machines are retrained quite often, we expect the algorithms to better adapt to different market regimes.

For comparing different machine learning algorithms, it is mandatory to ensure that all algorithms use exactly the same data, as well as avoiding look-ahead bias. Otherwise, the results would lack validity. For example, the look-ahead bias can be introduced during normalizing or scaling the data. If one seeks to use scaled or normalized data in a walk-forward environment, the scaling of the data has to strictly occur for each subsample separately. If one scales the whole dataset prior to the subsampling process and then applies a walk-forward routine, each subset will contain information which originally have not been available at that point in time. The reason is, that scaling and normalizing functions contain elements of order theory, e.g. global minima or global maxima, which depend on the sample at hand.

In our experiment we do not apply scaling, in order to keep the level of comparability among the algorithms as high as possible. This introduces a restriction regarding the activation function for our ANN algorithms, which will be addressed in Section 3.5. Moreover, as mentioned earlier, we removed all zero valued ROC before computing subsamples and inputs. This will hedge the risk of different treating of zero-values in our different algorithms, which could influence or bias the comparability. However, this will only affect our hourly datasets, as they contain considerably more zero-valued ROCs.

3.4 Performance Measures

The forecasting performance of our algorithms is evaluated by two types of performance measures. The first performance measure is the root-mean-square error (RMSE), which measures the magnitude of the deviation between a predicted value and the true observable value. A smaller RMSE indicates a closer prediction than a higher RMSE. Formally, we have:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \tag{3.2}$$

with y_t denoting the actual value and $\hat{y}_t$ the predicted value.

The other type of performance measures focuses on the directional quality of our predictions. We use the directional symmetry (DS) measure, which calculates the proportion of correctly predicted changes in direction from t_{i-1} to t_i from the total number of predictions.

$$DS = \frac{100}{n}\sum_{t=1}^{n} d_t \text{ with } d_t = \begin{cases} 1 & \text{if } (y_t - y_{t-1})(\hat{y}_t - \hat{y}_{t-1}) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

The higher the values of DS, the better is our model in predicting directional changes. Our third measure is the sign symmetry, which simply measures the percentage of total predictions that have the same sign as the actual values. Again, higher values of SS suggest a better predictions.

$$SS = \frac{100}{n}\sum_{t=1}^{n} s_t \text{ with } s_t = \begin{cases} 1 & \text{if } sign(y_t) = sign(\hat{y}_t) \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Both the DS and SS are rather important for traders as one of their goals in the first place, is to anticipate the correct direction of the next time step. Note that, even a very high value of DS and SS cannot guarantee sustainable profits or alternatively, a higher value of DS or SS does not necessarily result in a higher trading profit. Nevertheless, from an investors point of view, a higher DS or SS should be preferred, ceteris paribus.

3.5 Selected Hyper-Parameters

The importance of selecting adequate hyper-parameters for machine learning is manifold. Selecting improper parameters can result in over- and underfitting [40], which in return influences the generalization performance.

The methodology for finding the optimal set of parameters is an important, but yet not successively answered research question. Typical suggested methodologies are:

- Grid Search [48]
- Random Search [48]
- Bayesian Optimization [49, 50]
- Gradient Based Optimization [51]

As the emphasis of this thesis is to compare various machine learning algorithms and methods, we do not cover the optimization problems of our algorithms. Instead, we use similar hyper-parameters to those from related experiments.

In Table 3 we listed the selected hyper-parameters for our experiments. Note that, we decided to use a linear learning function for our ANN instead of a more commonly used sigmoid learning function. The reason for that is that the sigmoid learning function strictly requires scaled data in the range of $[0, 1]$ because it is only defined for $[0, 1]$.

Also, one should know that optimizing hyper-parameters with large datasets, i.e. financial time-series at hourly sampling rates, combined with a walk-forward routine, can easily become unfeasible for standard desktop multi-core central processing units. The same holds true for the required memory, which can easily exceed the standard 8 or 16 Gigabyte.

Algorithm	**Selected Hyper-Parameters**
SVM	
Kernel:	Radial basis function (Gaussian)
Cost	$C = 100$
Gamma	$\gamma = 0.1$
Epsilon	$\epsilon = 0.001$
Regression type	ϵ-regression
ANN	
Learning function	Standard Backpropagation
Learning parameter	$\eta = 0.1$
Hidden nodes	$size = 8$
Initiation function	Randomize weights
Activation function	Linear
***k*-NN**	
Number of neighbors	$k = 5$
KPCA	
Kernel:	Radial basis function (Gaussian)
Sigma:	$\sigma = 0.025$
Num. Features:	Features= 100
Ensemble	
Training divisor:	Divisor= 4
Iterations:	Iterations= 30

Table 3: The selected hyper-parameters for our machine learning algorithms

3.6 Research Questions

Before we move to the results of our experiments, we want to point out our key research questions, which we seek to answer. First, we analyze and compare the results for our three algorithms regarding their accuracy. Also, given various training window lengths and different numbers of OOS forecast per training, we search for the suitable values of the two parameters. This will be done on both, the DAX 30 and the S&P 500 datasets at daily and hourly time frames.

Next, we analyze the impact of KPCA, when conducted on the training data, before training our algorithms. We intend to find out if the KPCA can increase the forecasting performance of our algorithms, through reducing the dimensionality of the input data. Furthermore, we combine our algorithms with the bootstrap aggregation algorithm. More specifically, we investigate, if this ensemble method can outperform our single machines.

We close our experiments by providing backtesting and benchmarking results for a selection of trained machines. Therefore, we apply a simple trading decision method based on our machine's predictions and compare their performances with a buy-and-hold strategy in the respective index.

4 Results of the Experiment

In this Section, we present the results of our experiments. It is organized in Subsections according to our research questions.

4.1 Performance Comparison

4.1.1 Daily Data

The RMSEs of our three algorithms range from 0.015 to 0.024 in the DAX 30 dataset and slightly lower from 0.010 to 0.017 in the S&P 500 dataset. All machines achieve lower RMSEs in the S&P 500 dataset on average compared to the DAX 30 dataset. In Table 4 and 5 we provide the results for our three algorithms on the daily datasets.

The lowest RMSE for the DAX 30 dataset is achieved by the k-NN algorithm followed by the ANN with a slightly higher RMSE of 0.016, while the SVM ranks last with a higher RMSE of 0.019. The same order pertains for the highest RMSE. On our S&P 500 dataset, the ANN outperforms the k-NN by a 0.001 margin, while the SVM generates at least a RMSE of 0.013. It also produces the highest RMSE on the S&P 500 dataset.

Window	# OOS	RMSE SVM	DS SVM	SS SVM	RMSE ANN	DS ANN	SS ANN	RMSE KNN	DS KNN	SS KNN
100	10	0,021	50,262	49,213	0,021	59,288	51,536	0,016	53,109	49,700
100	25	0,021	49,944	48,710	0,019	58,879	50,729	0,016	55,402	50,953
100	50	0,021	50,604	49,434	0,020	61,321	49,509	0,016	58,906	48,642
200	20	0,021	50,625	49,492	0,018	59,570	48,945	0,016	52,188	49,922
200	50	0,021	51,686	49,412	0,019	61,804	51,412	0,017	54,000	50,784
200	100	0,021	50,760	49,560	0,018	59,240	49,680	0,016	58,680	51,400
400	40	0,021	51,653	51,653	0,017	61,949	52,712	0,017	50,847	49,492
400	100	0,021	51,609	51,087	0,018	60,478	48,783	0,017	51,043	49,826
400	200	0,021	52,727	51,136	0,017	63,682	50,045	0,017	59,364	50,773
800	80	0,022	50,781	50,990	0,018	62,031	50,885	0,018	51,771	51,250
800	200	0,022	50,167	49,111	0,019	65,278	50,722	0,019	52,500	49,389
800	400	0,024	49,563	48,625	0,020	61,438	50,063	0,019	59,875	49,625
1600	160	0,019	51,250	50,357	0,017	64,732	50,625	0,015	47,500	52,143
1600	400	0,019	50,250	50,125	0,016	61,000	49,000	0,016	52,625	53,750
1600	800	0,019	49,625	50,250	0,019	63,875	50,875	0,016	55,875	49,250
	Avg.	**0,021**	**50,767**	**49,944**	**0,018**	**61,638**	**50,368**	**0,017**	**54,246**	**50,460**
	Min	**0,019**	**49,563**	**48,625**	**0,016**	**58,879**	**48,783**	**0,015**	**47,500**	**48,642**
	Max	**0,024**	**52,727**	**51,653**	**0,021**	**65,278**	**52,712**	**0,019**	**59,875**	**53,750**

Table 4: Results of our three algorithms on the daily DAX 30 dataset for varying windows lengths and different numbers of OOS predictions per training

Window	# OOS	RMSE SVM	DS SVM	SS SVM	RMSE ANN	DS ANN	SS ANN	RMSE KNN	DS KNN	SS KNN
100	10	0,016	51,672	49,721	0,016	51,115	49,164	0,013	53,313	49,319
100	25	0,016	52,093	49,860	0,016	53,891	50,419	0,013	55,659	50,140
100	50	0,016	51,969	51,125	0,015	52,281	50,031	0,013	58,844	50,938
200	20	0,016	52,276	50,833	0,014	52,756	50,385	0,013	52,083	50,513
200	50	0,016	51,677	51,290	0,013	52,774	50,290	0,013	54,194	51,000
200	100	0,016	52,548	50,742	0,013	52,871	49,161	0,013	59,355	50,452
400	40	0,016	51,130	49,829	0,013	52,123	51,438	0,013	50,171	50,068
400	100	0,016	51,690	50,655	0,013	53,207	49,690	0,013	52,414	50,000
400	200	0,016	50,857	50,107	0,014	52,929	50,250	0,013	57,643	49,893
800	80	0,017	50,806	49,919	0,014	53,347	50,524	0,014	50,685	48,750
800	200	0,016	50,833	49,875	0,014	51,042	51,292	0,014	53,500	50,375
800	400	0,016	50,833	50,958	0,014	56,250	47,083	0,014	58,625	50,167
1600	160	0,013	53,125	51,438	0,011	53,500	48,938	0,011	51,313	51,188
1600	400	0,013	52,188	51,563	0,010	59,188	49,688	0,011	50,875	49,063
1600	800	0,013	51,813	52,750	0,011	59,750	53,688	0,011	57,188	49,500
	Avg.	**0,016**	**51,701**	**50,711**	**0,013**	**53,802**	**50,136**	**0,013**	**54,391**	**50,091**
	Min	**0,013**	**50,806**	**49,721**	**0,010**	**51,042**	**47,083**	**0,011**	**50,171**	**48,750**
	Max	**0,017**	**53,125**	**52,750**	**0,016**	**59,750**	**53,688**	**0,014**	**59,355**	**51,188**

Table 5: Results of our three algorithms on the daily S&P 500 dataset for varying windows lengths and different numbers of OOS predictions per training

Averaging over all 15 subsamples in each dataset, the k-NN achieves the lowest RMSEs on both indices, with an average 0.017 for the DAX 30, respectively 0.0126 on the S&P 500 data. The ANN is second, followed by the SVM.

Our direction performance measures show a different ranking. The highest DS levels are achieved by the ANN on the DAX 30 data with an average DS of 61.64. Whereas the k-NN holds the lead on the DAX data, but only by a small margin.

In terms of DS the SVM shows the worst average performance among the three. However, in predicting directional changes of the daily ROC all algorithms are able to be more often correct than wrong on both datasets.

In addition to that, our algorithms, with one exceptional case, are able to produce more correct predictions than incorrect predictions for the sign of the next day's index level change. The SVM has the highest average SS overall, which is achieved for the S&P 500. At the same time, the SVM is the one exception with a SS smaller than 50, with 49.43 for the DAX 30. The other two machines average levels around 50.40 for the DAX 30 and 50.10 for the S&P 500. Furthermore, the k-NN has the highest SS with 54.75 on the DAX 30 data, whereas the ANN obtains a peak SS of 53.69. In contrast, the k-NN ranks last in the other set.

Considering the average scores of all three performance measures for our algorithms, across both datasets the ANN and the k-NN outperform the SVM under the vast majority of cases.

4.1.2 Hourly Data

The results of our algorithms for hourly data are depicted in Tables 6 and 7. Again, we ascertain smaller levels of RMSEs for the S&P 500 than for the DAX 30. Overall, the RMSEs are smaller than in the daily dataset, which is due to the shorter time frame. The mean and standard deviation of the hourly data are lower than for the daily data, see Figure 1.

The best RMSEs on both datasets are achieved by the k-NN, considering the average, minimum and maximum RMSE values. Second, we have the ANN, while the SVM produces the highest RMSEs again. In contrast to the daily data, we have a clear cut order regarding the RMSEs of our three learning machines.

The DS of our algorithms on the hourly data are lower on average than for the daily dataset, except for the SVM, which has slightly higher average DS on hourly data. Even the minimums of the SVM are greater than 50. The k-NN produces the best average DS values with 52.9995 (S&P 500) and 53.9754 (DAX 30) while the highest average SS values are obtained by the SVM with 50.6663 (DAX 30) and

Window	# OOS	RMSE SVM	DS SVM	SS SVM	RMSE ANN	DS ANN	SS ANN	RMSE KNN	DS KNN	SS KNN
800	80	0,0054	50,6167	50,4075	0,0044	52,1090	50,3442	0,0042	50,3524	50,0909
800	200	0,0054	50,4945	50,3923	0,0045	52,2238	50,1713	0,0042	52,1492	50,1188
800	400	0,0053	50,5611	50,6194	0,0044	51,7861	49,7694	0,0042	57,3389	49,7333
1600	160	0,0051	51,0586	50,8080	0,0043	52,2889	50,3266	0,0042	50,4730	49,9352
1600	400	0,0051	51,0795	50,8608	0,0043	53,0597	50,5710	0,0042	51,8438	50,1875
1600	800	0,0051	51,1818	50,7301	0,0043	52,3551	50,3352	0,0042	57,3949	50,1648
3200	320	0,0050	51,4239	50,9876	0,0043	51,8190	50,4127	0,0042	50,2889	50,0943
3200	800	0,0049	51,5387	50,7649	0,0043	51,8839	50,1161	0,0042	51,2560	50,1964
3200	1600	0,0048	51,2589	50,5893	0,0042	53,8780	50,5744	0,0042	57,0268	50,1042
6400	640	0,0049	51,4193	50,3418	0,0045	51,8620	50,3320	0,0044	49,9772	50,0260
6400	1600	0,0049	51,7072	50,4178	0,0044	54,1875	50,7961	0,0043	51,8553	49,9605
6400	3200	0,0050	51,8819	50,4757	0,0047	51,9514	48,9757	0,0044	56,5868	50,3542
12800	1280	0,0050	52,6974	50,8840	0,0048	50,0699	49,7738	0,0047	50,0164	50,0247
12800	3200	0,0051	52,5357	50,9598	0,0049	54,8304	50,6116	0,0047	51,7545	50,6250
12800	6400	0,0053	52,9219	50,7552	0,0050	54,9323	49,4375	0,0050	56,3177	49,5885
	Avg.	**0,0051**	**51,4918**	**50,6663**	**0,0045**	**52,6158**	**50,1698**	**0,0044**	**52,9754**	**50,0803**
	Min	**0,0048**	**50,4945**	**50,3418**	**0,0042**	**50,0699**	**48,9757**	**0,0042**	**49,9772**	**49,5885**
	Max	**0,0054**	**52,9219**	**50,9876**	**0,0050**	**54,9323**	**50,7961**	**0,0050**	**57,3949**	**50,6250**

Table 6: Results of our three algorithms on the hourly DAX 30 dataset for varying windows lengths and different numbers of OOS predictions per training

50.5848 (S&P 500). Furthermore, all algorithms can correctly predict the sign of the next hour's logarithmic return more often than every second time, meaning their average SS exceed 50.

4.2 Optimal Training Window lengths

We increased the training length by factor two, for four times, starting at 100 days for the daily datasets and 800 hours for the hourly datasets. The longest training windows contain 1.600 days, respectively 12.800 hours. Tables 4, 5, 6 and 7 show all results for the different training window lengths.

4.2.1 Daily Data

All learning machines reach their lowest RMSEs on the longest training window consisting of 1.600 training observations. However, the k-NN is able to obtain low levels of RMSEs already from the smallest training window of 100 days. It can accomplish the minimum RMSE of the SVM on 100 training observations, while the SVM needs 1.600 observations. In most cases, the RMSE improves when training length is increased.

Window	# OOS	RMSE SVM	DS SVM	SS SVM	RMSE ANN	DS ANN	SS ANN	RMSE KNN	DS KNN	SS KNN
800	80	0,0036	51,0276	50,3005	0,0031	50,8184	50,1544	0,0028	50,3254	50,2822
800	200	0,0036	50,9518	50,4585	0,0034	50,4684	49,8322	0,0028	52,1927	50,0797
800	400	0,0036	51,0067	50,3483	0,0031	50,7533	49,7817	0,0028	57,6533	50,2717
1600	160	0,0035	51,1540	50,4751	0,0031	51,0478	50,0623	0,0029	50,1567	50,2375
1600	400	0,0035	51,1740	50,5997	0,0029	51,0743	50,0811	0,0029	51,3970	49,5777
1600	800	0,0034	50,9375	50,3615	0,0030	50,1520	49,8176	0,0028	57,3699	50,3209
3200	320	0,0034	51,2830	50,9479	0,0029	50,1979	50,4896	0,0029	50,1389	50,0556
3200	800	0,0033	51,4531	50,8785	0,0029	51,1250	49,7569	0,0029	51,8542	50,0503
3200	1600	0,0033	51,4080	50,7622	0,0030	49,2865	50,2257	0,0029	57,0677	50,2257
6400	640	0,0033	51,8070	50,5846	0,0029	52,8051	50,0129	0,0029	50,0919	50,3548
6400	1600	0,0033	51,6011	50,5790	0,0030	50,2096	50,6305	0,0029	51,2831	49,7794
6400	3200	0,0033	51,4265	50,7574	0,0030	50,6250	50,0533	0,0029	56,6434	49,7279
12800	1280	0,0034	52,7724	50,5617	0,0030	52,2530	49,7635	0,0031	49,8628	50,1098
12800	3200	0,0033	52,6875	50,5104	0,0033	52,5563	49,8375	0,0031	51,8375	50,2917
12800	6400	0,0034	53,2679	50,6473	0,0033	53,0536	50,2254	0,0031	57,1183	49,9353
	Avg.	**0,0034**	**51,5972**	**50,5848**	**0,0031**	**51,0951**	**50,0483**	**0,0029**	**52,9995**	**50,0867**
	Min	**0,0033**	**50,9375**	**50,3005**	**0,0029**	**49,2865**	**49,7569**	**0,0028**	**49,8628**	**49,5777**
	Max	**0,0036**	**53,2679**	**50,9479**	**0,0034**	**53,0536**	**50,6305**	**0,0031**	**57,6533**	**50,3548**

Table 7: Results of our three algorithms on the hourly S&P 500 dataset for varying windows lengths and different numbers of OOS predictions per training

The DS and SS do not show a systematic pattern for the different training window lengths, as the levels of DS and SS fluctuate unsteadily.

4.2.2 Hourly Data

Analyzing the RMSEs of our algorithms for hourly data, we observe that increasing the training window length improves the RMSE only until training windows lengths reach 3.200 or 6.400 observations. Advancing to 12.800 frequently cause the RMSEs of our learning machines to steepen up to their average RMSEs or higher. This holds particularly true for the k-NN algorithm, which validates its capability to produce low levels of RMSEs from short training window lengths as against the SVM and ANN, as well on hourly data.

The SVM appears to achieve higher levels of DS systematically when training windows length is raising. In both datasets it reached its maximum DS, when trained with 12.800 observations, paired with average SS. For the other two algorithms, there are no substantial dynamics in the DS and SS measures.

4.3 Optimal Number of OOS Predictions

In order to gain an intuition about how many OOS predictions to produce before retraining, we choose three different testing lengths, which always are 10%, 25% and 50% of the training window lengths.

Considering all daily and hourly datasets, we observe that variations in the number of OOS predictions only have a marginal impact on the RMSE of our algorithms. This can be concluded by keeping the training window length constant and analyzing the variations in RMSEs for the 10%, 25% and 50% OOS proportions of training length.

Our experiments cannot highlight a certain proportion, which systematically outperforms other proportions across all datasets and algorithms.

4.4 Kernel Principle Component Analysis

Next, we examine if the application of KPCA can further improve the forecasting performance of the algorithms. In Table 8 we compare the predicting performance of our previous learning machines to the predicting performance of the same machines, when KPCA was conducted to the input data, prior to each training session of every algorithm. The selected training window lengths are 1.600 days for the daily sets, respectively 3.200 hours for the hourly sets.

The RMSEs of our learning algorithms decrease or remain close to their previous RMSEs in the vast majority of cases, except for the ANN. This hold particularly true for the SVM, as its RMSEs decrease in all cases through KPCA, while for the k-NN we observe the RMSEs remain constant or slightly increasing occasionally. Our results exhibit a different behavior of the KPCA-ANN. Here the RMSEs mostly rise or stay close the previous values when KPCA is conducted.

However, examining the DS after applying KPCA reveals very large enhancements for the SVM and the k-NN. DS values higher than 80 or even 90 are produced systematically from both learning machines, when trained on the extracted features. Divisively, their SS usually fall, when DS levels increase by great margins.

In contrast, we can see a different results for the ANN with KPCA regarding

	Window	**# OOS**	**RMSE SVM**	**DS SVM**	**SS SVM**	**RMSE ANN**	**DS ANN**	**SS ANN**	**RMSE KNN**	**DS KNN**	**SS KNN**
	DAX 30 daily										
	1600	160	0,0190	51,25	50,36	0,0169	64,73	50,63	0,0152	47,50	52,14
KPCA	1600	160	0,0190	56,43	48,66	0,0141	54,11	49,20	0,0148	49,82	51,52
	1600	400	0,0191	50,25	50,13	0,0161	61,00	49,00	0,0162	52,63	53,75
KPCA	1600	400	0,0171	52,38	47,50	0,0171	51,50	50,38	0,0165	53,38	48,63
	1600	800	0,0194	49,63	50,25	0,0188	63,88	50,88	0,0162	55,88	49,25
KPCA	1600	800	0,0152	53,13	48,63	0,0268	55,25	54,63	0,0157	59,75	54,13
	S&P 500 daily										
	1600	160	0,0132	53,13	51,44	0,0107	53,50	48,94	0,0106	51,31	51,19
KPCA	1600	160	0,0095	99,56	48,38	0,0114	51,69	51,00	0,0098	73,56	49,56
	1600	400	0,0132	52,19	51,56	0,0103	59,19	49,69	0,0109	50,88	49,06
KPCA	1600	400	0,0095	99,88	49,00	0,0119	52,50	52,00	0,0099	81,69	49,75
	1600	800	0,0128	51,81	52,75	0,0113	59,75	53,69	0,0107	57,19	49,50
KPCA	1600	800	0,0095	99,94	45,00	0,0112	47,31	46,13	0,0100	85,31	49,31
	DAX 30 hourly										
	3200	320	0,0050	51,42	50,99	0,0043	51,82	50,41	0,0042	50,29	50,09
KPCA	3200	320	0,0039	99,83	50,93	0,0042	50,46	49,64	0,0043	53,73	49,80
	3200	800	0,0049	51,54	50,76	0,0043	51,88	50,12	0,0042	51,26	50,20
KPCA	3200	800	0,0039	99,93	50,87	0,0043	50,27	50,30	0,0043	56,39	50,27
	3200	1600	0,0048	51,26	50,59	0,0042	53,88	50,57	0,0042	57,03	50,10
KPCA	3200	1600	0,0039	99,96	51,09	0,0042	50,71	50,30	0,0043	62,77	50,08
	S&P 500 hourly										
	3200	320	0,0034	51,28	50,95	0,0029	50,20	50,49	0,0029	50,14	50,06
KPCA	3200	320	0,0027	99,84	50,36	0,0030	49,11	50,13	0,0027	99,79	50,18
	3200	800	0,0033	51,45	50,88	0,0029	51,13	49,76	0,0029	51,85	50,05
KPCA	3200	800	0,0027	99,93	50,24	0,0029	49,22	50,49	0,0028	99,91	50,38
	3200	1600	0,0033	51,41	50,76	0,0030	49,29	50,23	0,0029	57,07	50,23
KPCA	3200	1600	0,0027	99,97	50,04	0,0030	50,40	50,47	0,0028	99,93	50,32

Table 8: Results of the KPCA compared to the previous results

	Window	# OOS	RMSE SVM	DS SVM	SS SVM	RMSE ANN	DS ANN	SS ANN	RMSE KNN	DS KNN	SS KNN
	DAX 30 daily										
	1600	160	0,0190	51,25	50,36	0,0169	64,73	50,63	0,0152	47,50	52,14
Bagging	1600	160	0,0143	52,77	51,88	0,0134	55,89	50,63	0,0133	50,27	47,77
	1600	400	0,0191	50,25	50,13	0,0161	61,00	49,00	0,0162	52,63	53,75
Bagging	1600	400	0,0159	49,88	49,63	0,0149	59,13	48,13	0,0145	45,88	48,13
	1600	800	0,0194	49,63	50,25	0,0188	63,88	50,88	0,0162	55,88	49,25
Bagging	1600	800	0,0156	50,50	49,25	0,0146	57,25	48,63	0,0145	50,38	51,25
	S&P 500 daily										
	1600	160	0,0132	53,13	51,44	0,0107	53,50	48,94	0,0106	51,31	51,19
Bagging	1600	160	0,0104	51,63	51,06	0,0097	57,00	48,00	0,0095	51,00	52,69
	1600	400	0,0132	52,19	51,56	0,0103	59,19	49,69	0,0109	50,88	49,06
Bagging	1600	400	0,0105	51,63	50,13	0,0096	58,31	49,88	0,0096	49,69	49,25
	1600	800	0,0128	51,81	52,75	0,0113	59,75	53,69	0,0107	57,19	49,50
Bagging	1600	800	0,0104	51,31	50,69	0,0097	61,81	47,50	0,0095	50,19	49,06
	DAX 30 hourly										
	3200	320	0,0050	51,42	50,99	0,0043	51,82	50,41	0,0042	50,29	50,09
Bagging	3200	320	0,0041	51,59	50,86	0,0039	50,69	50,78	0,0039	50,05	50,06
	3200	800	0,0049	51,54	50,76	0,0043	51,88	50,12	0,0042	51,26	50,20
Bagging	3200	800	0,0041	51,76	50,93	0,0040	47,88	50,18	0,0039	49,93	50,77
	3200	1600	0,0048	51,26	50,59	0,0042	53,88	50,57	0,0042	57,03	50,10
Bagging	3200	1600	0,0041	51,08	50,88	0,0040	50,78	49,79	0,0039	49,84	50,73
	S&P 500 hourly										
	3200	320	0,0034	51,28	50,95	0,0029	50,20	50,49	0,0029	50,14	50,06
Bagging	3200	320	0,0028	51,21	50,78	0,0027	52,31	50,38	0,0027	49,93	50,19
	3200	800	0,0033	51,45	50,88	0,0029	51,13	49,76	0,0029	51,85	50,05
Bagging	3200	800	0,0028	51,35	50,68	0,0027	53,21	50,74	0,0027	49,90	50,22
	3200	1600	0,0033	51,41	50,76	0,0030	49,29	50,23	0,0029	57,07	50,23
Bagging	3200	1600	0,0028	51,26	50,54	0,0027	52,37	50,38	0,0027	49,86	49,95

Table 9: Results of the bootstrap aggregating algorithms ("Bagging") compared to the previous results

their DS and SS. In nearly all cases the DS decrease in utilization of KPCA while the SS fluctuate bidirectionally, revealing no systematic pattern. Solely valid by some exceptions, we notice that increases in SS often come along with increases in RMSEs and vice versa. Again, there are multiple exceptions for this rule in our data.

4.5 Bootstrap Aggregating

Finally, we employ an ensemble method, namely the bootstrap aggregating algorithm. As for the KPCA, we use the same training window lengths that yielded better performance results relative to the other window lengths. Table 9 summarizes the results of our ensemble method.

Through the utilization of bootstrap aggregating the RMSEs of all learning machines reduced in all of our datasets. Bootstrap aggregation could reduce the RMSEs of our three initial algorithms by 0.0023 on average on our daily DAX 30 and S&P 500 datasets. In our hourly datasets we achieve a difference of 0.00038 on average.

The reductions in RMSEs come along with fluctuations in DS and SS, upwards as well as downwards varying in magnitude. The magnitudes of the fluctuations in DS and SS are the lowest for the SVM in both daily and hourly data with average differences ranging from 0.02 to 0.64. This represents a slight worsening through bootstrap aggregating.

Our ensembles of ANN and k-NN produce changes of greater magnitude in DS in both directions. For instance, the DS of our ANN decrease a lot in both DAX 30 datasets, while the DS rather increase on the S&P 500 datasets. The DS of the ensemble k-NN is significantly reduced through all datasets, except for one out of 12 cases. In contrast, the SS increased slightly in seven cases.

4.6 Backtesting Results for our Trading Systems

In order to provide an impression on the profitability of our machine learning algorithms, we embed forecasts generated by our algorithms in a simple trading system, which we thereafter simulate through our datasets. Based on the sign of the one-step ahead ROC prediction, our system decides to be either fully invested in a long trade or in a short trade for the next time step. It keeps the position until a change in the sign of the next period prediction is forecasted. In this case, it instantly liquidates the position and opens another according to the new predicted direction. Of course, this is not a proper trading system, nor a valid backtesting approach. Both contain multiple shortcomings. However, it is beyond scope of this thesis to delve deep into algorithmic trading. The results are presented solely for the purpose of providing an impression of the profitability.

We compute multiple performance measures[2] for a selection of our algorithms, which are listed in Table 10. We selected two algorithms, one for the DAX 30 and one for the S&P 500, which performed well in our previous experiments in terms of performance measures. Especially a high level of SS is important for our trading

[2]Measures calculated according to [52]

	1	2	3	4	5	6
Dataset	Daily Dax 30	Daily S&P 500	Daily Dax 30	Daily Dax 30	Daily Dax 30	Daily Dax 30
Window/OOS	1600/400	1600/800	1600/800	1600/800	1600/160	1600/160
Algorithm	kNN	ANN	k-NN	k-NN	SVM	SVM
KPCA/Bagging	None	None	KPCA	None	Bagging	None
Cumulative Return	1,310	0,125	0,931	0,003	-0,231	-0,269
Annualized Return	0,302	0,019	0,230	0,001	-0,057	-0,068
Annualized Sharpe-Ratio	1,313	0,125	1,001	0,004	-0,273	-0,324
Annualized Std. Dev.	0,538	0,514	0,542	0,493	0,511	0,504
Max Drawdown	-0,206	-0,285	-0,210	-0,319	-0,510	-0,400
Max Length Drawdown	166	896	279	501	990	990
Cumulative Return	0,271	1,065	0,271	0,271	0,532	0,532
Annualized Return	0,078	0,121	0,078	0,078	0,101	0,101
Annualized Sharpe-Ratio	0,340	0,802	0,340	0,340	0,480	0,480
Annualized Std. Dev.	0,546	0,550	0,546	0,546	0,546	0,546
Max Drawdown	-0,334	-0,215	-0,334	-0,334	-0,334	-0,334
Max Length Drawdown	509	265	509	509	509	509

Table 10: Backtesting results for selected learning machines (top) compared to a buy-and-hold strategy in the respective index (bottom)

systems. According to the same principle, we choose one algorithm from the KPCA experiment and one learning machine which utilized bootstrap aggregating. Then, we pick the same two learning algorithms again, but without the KPCA - and bootstrap aggregating respectively for the sake of comparison.

Among our selected algorithms, the best performance was achieved by the k-NN algorithm (1) with the highest annualized risk adjusted returns, measured by the famous Sharpe-Ratio. It clearly outperformed the DAX buy-and-hold strategy in every measure. Comparing algorithms 3 and 4 reveals that the application of KPCA turned the poor algorithm (4) into a way more profitable algorithm (3), which enabled the algorithm to clearly outperform the DAX 30. However, the bootstrap aggregating SVM (5) even worsened the performance of the standard SVM (6). We enclosed performance charts of algorithms (1-6) and two additional SVMs trading the hourly DAX 30 and S&P 500 in the Appendix.

5 Discussion

In our performance comparison for the daily data, the SVM brought out the poorest predictions among the three learning algorithms in terms of average RMSEs. Both, the k-NN and the ANN produce lower RMSEs on average. Regarding the SS, we found the SVM to produce stable results considering levels of SS. Partially, our results contradict with previous related experiments, which merely found the SVM to be superior over k-NNs and ANNs on daily data by the RMSE, or similar squared error measures. This may be due to the usage of different input variables and hyper-parameters and methodological deviations from our approaches. We believe, the latter plays a greater role, as related experiments often made use of scaling and smoothed output variable in advance, that can easily affect comparability of the machines, if not conducted with high diligence. We clarified, that scaling has to take place before training to avoid look-ahead bias. If the entire data is scaled in advance and then subsampled, every decision made by a learning system would be based on data comprising sure information about the future. That would, non-controversially, break the rules for financial-time series forecasting. Related papers, usually do not provide detailed information on their scaling approaches, which may be due to unawareness of this pitfall. However, it must be explained in detail at least for the sake of reproducibility.

Through the analysis of optimal window lengths, we found that the performance of our learning machines depend on the training window length to some extend. For daily data, the longest training window length under study of 1.600 yielded the best results. In the hourly datasets, the best performances were achieved with 3.200 - 6.400 training observations. Overall, for the purpose of financial time series forecasting with machine learning algorithms, we recommend to use at least 1.000 training observations, as shorter training windows tend to produce inferior performances. We find this particularly relevant with regard to SVMs and ANNs. The k-NN method approaches its maximum potential faster, meaning it needs less observations to come close to its best performances.

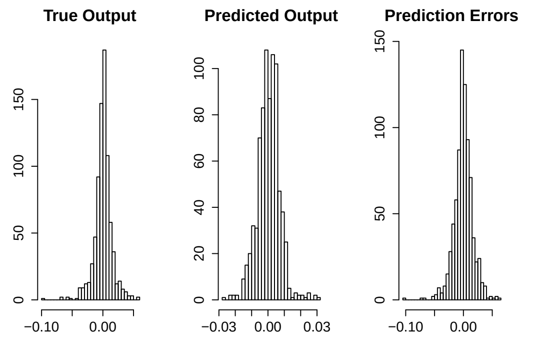
The high values of DS achieved through the application of KPCA shed light on a weakness of this performance measure in its calculation. We recall (3.3):

$$DS = \frac{100}{n} \sum_{t=1}^{n} d_t \text{ with } d_t = \begin{cases} 1 & \text{if } (y_t - y_{t-1})(\hat{y}_t - \hat{y}_{t-1}) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

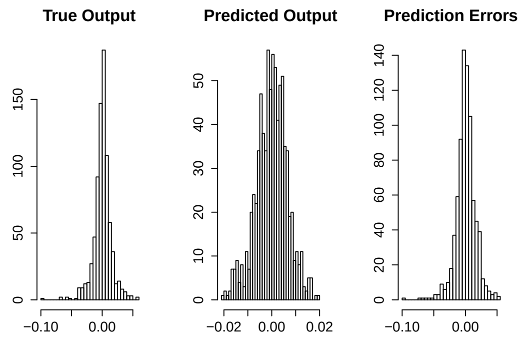
The weakness origins from term d_t which will take the value of one if $(y_t - y_{t-1})(\hat{y}_t - \hat{y}_{t-1})$ is greater or equal than zero, meaning that the change in direction was correctly predicted. Note that, $(y_t - y_{t-1})(\hat{y}_t - \hat{y}_{t-1})$ will equal zero if either $(y_t - y_{t-1})$ or $(\hat{y}_t - \hat{y}_{t-1})$ equals zero. Hence, if the actual value y or the predicted value $\hat{y}$ does not change at all in value from $t-1$ to t, then d_t will judge that as a correctly predicted directional change, which is wrong in any different case than for: $(y_t - y_{t-1}) = 0$ and $(\hat{y}_t - \hat{y}_{t-1}) = 0$. Thus the DS will increase in some situations, when it actually should decrease.

A strange coincidence is that in an earlier paper, comparing different algorithms for financial time series forecasting, Cao and Tay (2001) [1] utilize the DS and three other measures. As well in a later paper from 2003, again in the domain of SVMs for financial time-series forecasting [40]. In a third paper [43], Cao et al.(2003) analyzed the impact of KPCA for SVMs on financial time series data, among other datasets, however this time, only measured by the NMSE. They summarized, that KPCA can improve generalization performance, which we cannot confirm without limitations. Formally, a lower MSE through KPCA endorses this thesis, but it somehow hides an undesirable effect - the strong bias introduced to the predicting machine. In Figure 12 we show that a lower RMSE through KPCA may not generally be preferable for financial forecasting, containing two different examples in which the utilization of KPCA reduced the RMSE. In Figures 12a and 12b we observe that the histograms of the predicted outputs remain quite symmetric, containing predictions with positive and negative values. However, in the negative example in Figures 12c and 12d we notice that after the application of KPCA the predicted outputs only contain two classes of outputs, while both classes being negative in value.

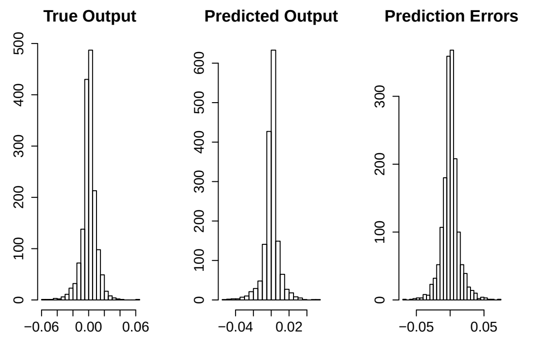
Clearly, this is not a desirable feature for a predicting machine in financial time series forecasting, as this particular machine becomes useless, as it only contains



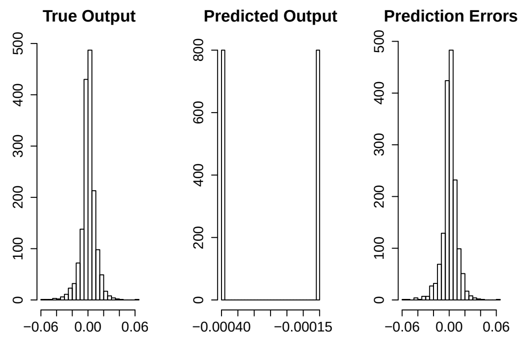
(a) Positive Example: k-NN before KPCA (RMSE=0.0162)



(b) Positive Example: k-NN with KPCA (RMSE=0.0157)



(c) Negative Example: SVM before KPCA (RMSE=0.0128)



(d) Negative Example: SVM with KPCA (RMSE=0.0095)

Figure 12: Histograms before and after utilization of KPCA for the true output,the predicted output and the prediction error

negative predictions. One could argue that it produced lower RMSEs, but in financial time series forecasting we also care about the correct direction of a forecast. In the negative example we see that the SVM with KPCA only models the left half of the true output histogram. If one only seeks to minimize the RMSE of a prediction, one can simply use the training sample mean of the output variable as a constant predictor, which is very likely to produce a low RMSE. Additionally, this predictor is certain to obtain a DS of 100. That leads to the finding that the application of

KPCA should not be evaluated by the RMSE only, in the domain of financial time series forecasting. In this example, the wrong interpretation of RMSE could have been prevented if the SS measure had been taken into consideration. In the positive example the SS values increased through the KPCA and vice versa in the negative example.[3]

Our bootstrap aggregation algorithms could produce lower RMSEs than the single learning machines. In contrast to the KPCA, it did not introduce the aforementioned undesirable features. However, bagging gave both, improvements and worsening in SS. However, the fluctuations are usually small. Considering the reductions in RMSE and the small changes in SS, we found the bootstrap aggregation helpful for forecasting financial time series forecasting. Nevertheless one has to take into account that bagging introduces new free parameters to the learning problem, which have to be determined by the supervisor in the optimization process. For the objective of improving a learning machine's generalization performance, we prefer bootstrap aggregating over KPCA.

We believe our trading systems to rather represent a lower bound on the general profitability potential. First, we did not optimize our algorithms and nor we did not try different input variables. Second, our trading strategy is quite naive and omits risk- and money-management. Further, in a real-world application, one must not require the machine to be invested with every prediction it produces. However, we have not taken bid/ask spreads and transactions costs into account which both would diminish the trading performance in a real-world application to some extend.

6 Conclusion and Outlook

The aim of this thesis has been to apply and compare different machine learning methods for forecasting non-linear financial time series. Further, optimal training window lengths and the impact of KPCA and the Bootstrap Aggregating Algorithm have been research questions. Before we conducted our experiments, we first

[3]In the positive example, we used the k-NN 1.600/800 DAX 30 daily from our experiments. In the negative example, we used the SVM 1.600/800 S&P 500 daily.

provided a theoretical introduction to the field of machine learning and three prominent learning algorithms for classification and regression, namely SVMs, k-NNs and ANNs.

During our experiments we measured the performance of our algorithms by RMSE, DS, and SS. Our research revealed that the DS measure has a general weakness in calculation, which can misguide interpretations significantly. Regarding the RMSE, the k-NN and the ANN outperformed the SVM in most cases, whereas the SVM produced the most stable performance in terms of SS. For forecasting daily directions, a training window length 1.600 days yielded lower levels of RMSE than shorter training windows. For hourly predictions, training window lengths of 3.200 and 6.400 hours were preferable to shorter and longer training window lengths. Overall, we realized that all of the three algorithms may be useful in forecasting financial time series, since the levels of SS were usually higher than 50. Additionally we investigated the influence of KPCA, when applied to the learning algorithms in training. Results showed that it often reduced RMSEs, while the effect on the SS was ambiguous. Additionally, the decrease in RMSE was often accompanied with the introduction of a strong bias, exposed by DS approaching 100. Summarizing KPCA can be helpful for financial forecasting, when handled with caution. It is insufficient to evaluate the influence of KPCA only by RMSEs in the domain of financial time series forecasting. Through bootstrap aggregating, the RMSEs of our learning algorithms could be reduced as well, while it introduced fluctuations in DS and SS in both directions, which were less for the SVM compared to the k-NN and the ANN. As the SVM provided the most stable SS values previously, we found bootstrap aggregating particularly useful when applied to stable standalone algorithms. Though, as for the KPCA, it is indispensable to evaluate this method by multiple performance measures.

Due to computational restraints, our experiments were limited in the selection of hyper-parameters and more sophisticated hybrid ensembles introducing new free parameters subject to optimization. Since our experiments did not cover the optimization of parameters, which is led for future research, we used recommendations from previous works instead.

We are convinced that machine learning methods will continue to be applied and further researched for the purpose of forecasting financial time series. In our opinion, methods have to be extended by further elements in the future, for which we provide a few ideas in the following. In fact, it could be interesting to train a machine only with observations where the output variable is outside a threshold channel, containing an upper- and a lower-bound threshold. It may be possible that the machine does better on predicting large (more profitable) movements, when trained on filtered data. Furthermore, one should keep in mind that in case of news releases with great impact, a trained machine based on technical indicators is not likely to predict the large movement on the next day or hour, because news cannot be anticipated by lagged technical indicators. Hence, a machine will not be able to learn from this patterns. Instead it will produce large errors. In the best case, a training dataset should contain only data of certain regimes of the stock market, which show similar input-output relationships. Or, in other words, the training data should not contain large movements of the output variable, which one believes to origin from news releases, that cannot be explained by the input variables at all. However, this machine would only be able to give good predictions in certain regimes.

Therefore, it could be possible to benefit of cross sectional relationships by using multiple sources of data. For instance, combining technical indicators, economic variables and sentiments from social media channels. That could help the machine predicting large movements, which are not based on technical indicators.

Of course, computational cost remains a large factor limiting standard computer systems, especially for the inevitable optimization process. This issue is already addressed by the field of graphical processing unit accelerated computing (GPU-accelerated computing), which exploits the high parallelization power of GPUs. However, GPU-accelerated computing is still in a very early stage of research.

Appendix

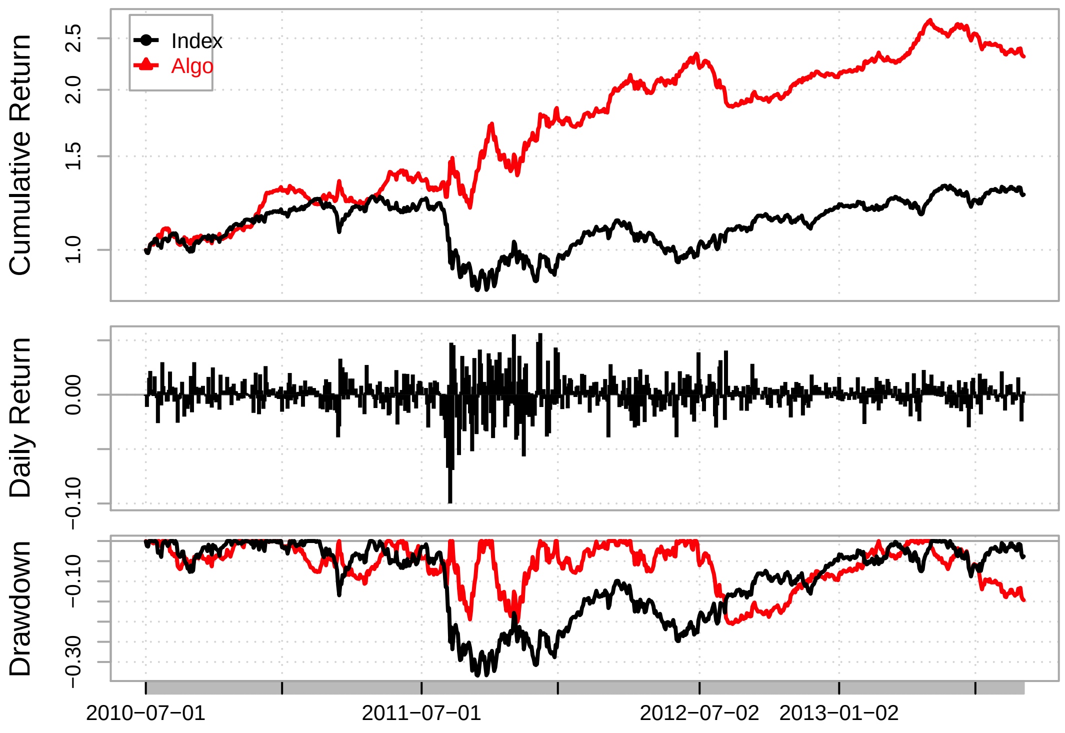


Figure 13: Performance Benchmark Charts for: Standard *k*-NN, 1600 window length, 400 OOS, DAX 30 daily data

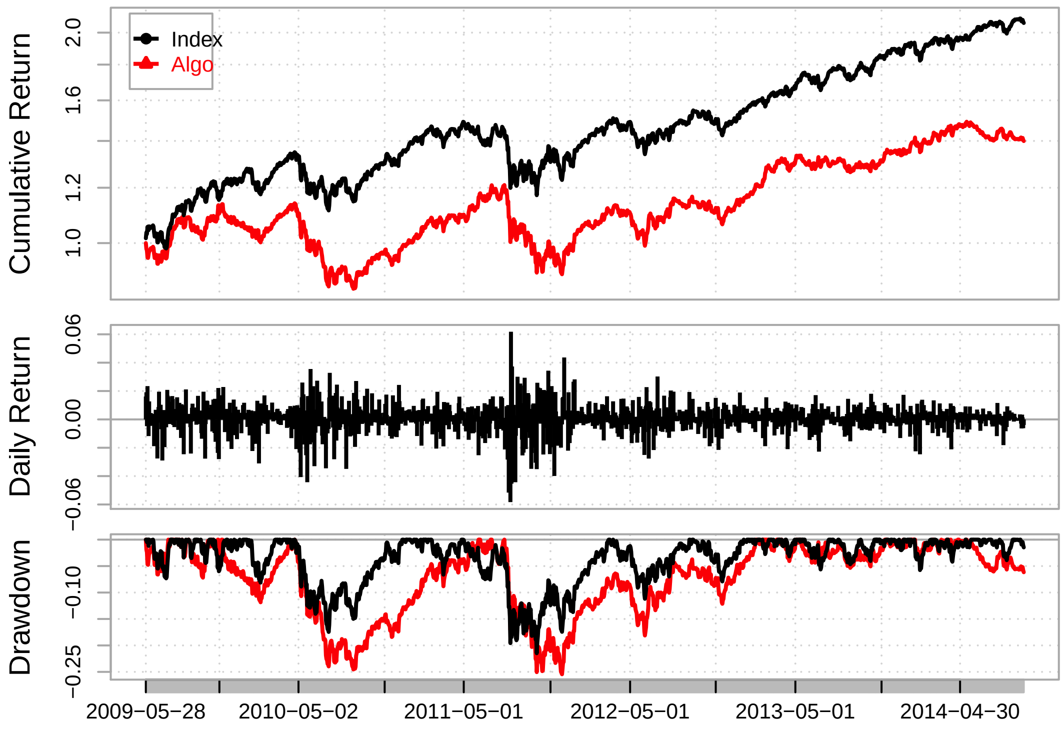


Figure 14: Performance Benchmark Charts for: Standard ANN, 1600 window length, 800 OOS, S&P 500 daily data

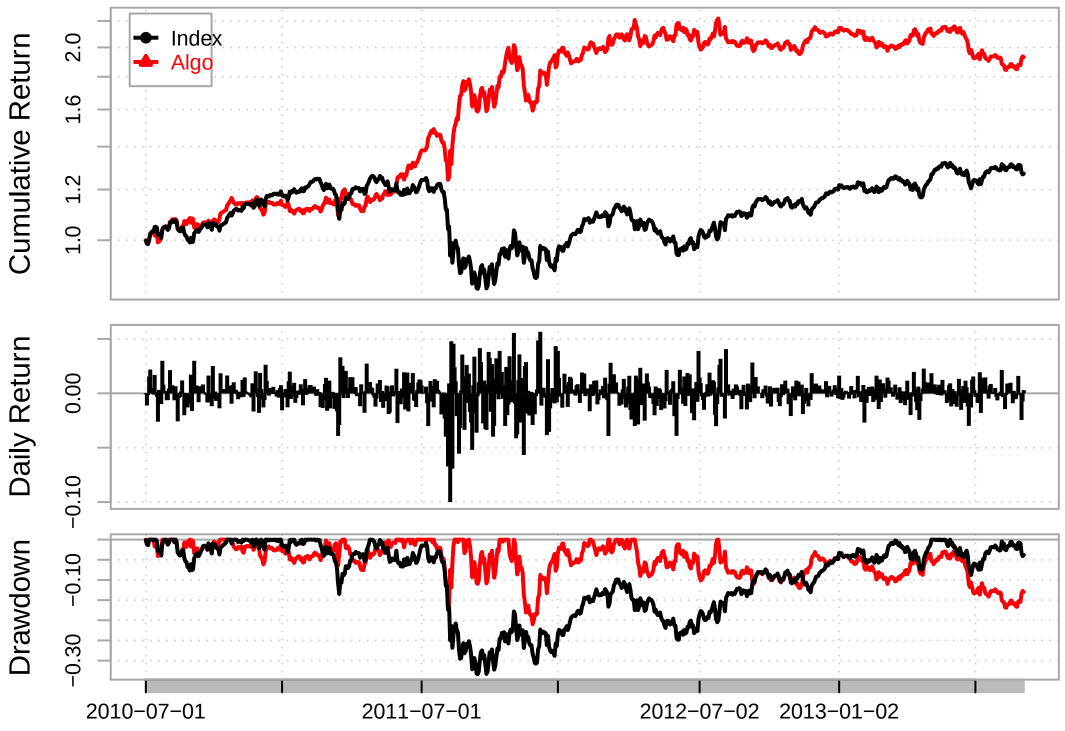


Figure 15: Performance Benchmark Charts for: KPCA k-NN, 1600 window length, 800 OOS, DAX 30 daily data

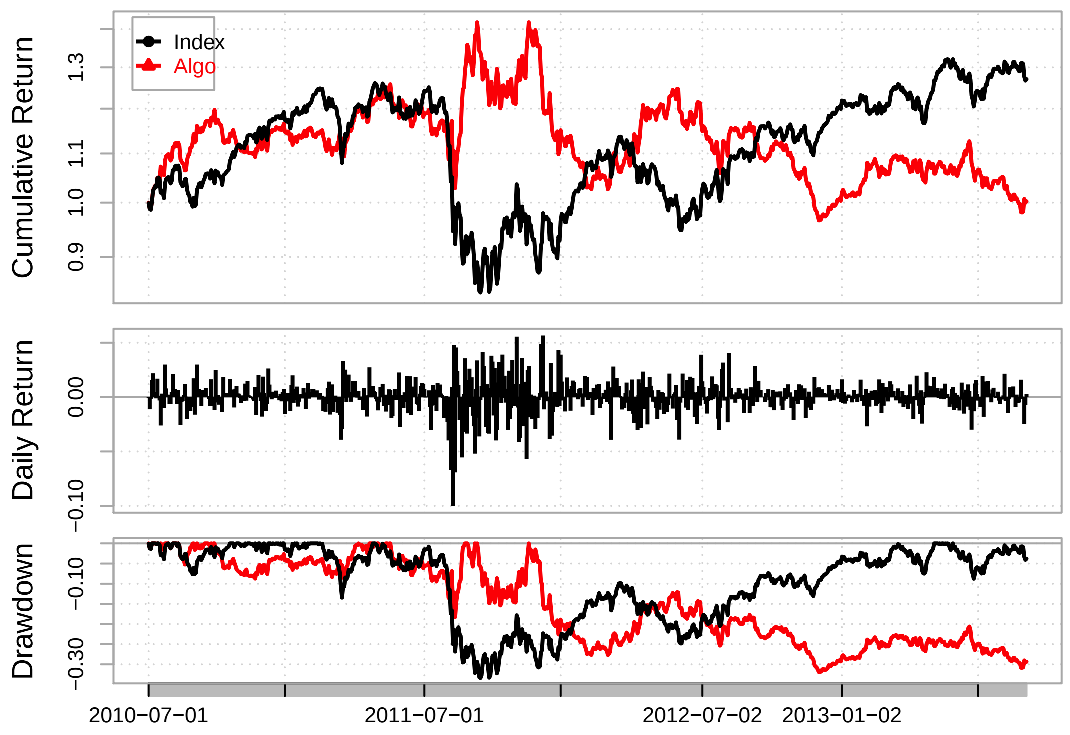


Figure 16: Performance Benchmark Charts for: Standard k-NN, 1600 window length, 800 OOS, DAX 30 daily data

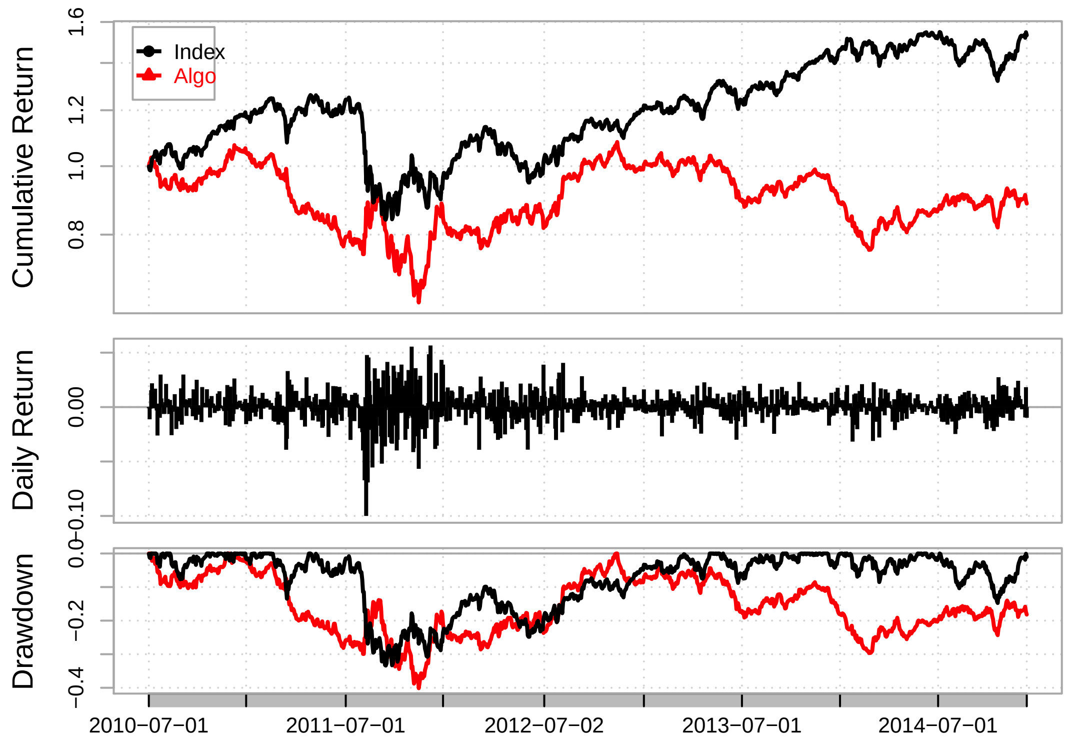


Figure 17: Performance Benchmark Charts for: Bagging SVM, 1600 window length, 160 OOS, DAX 30 daily data

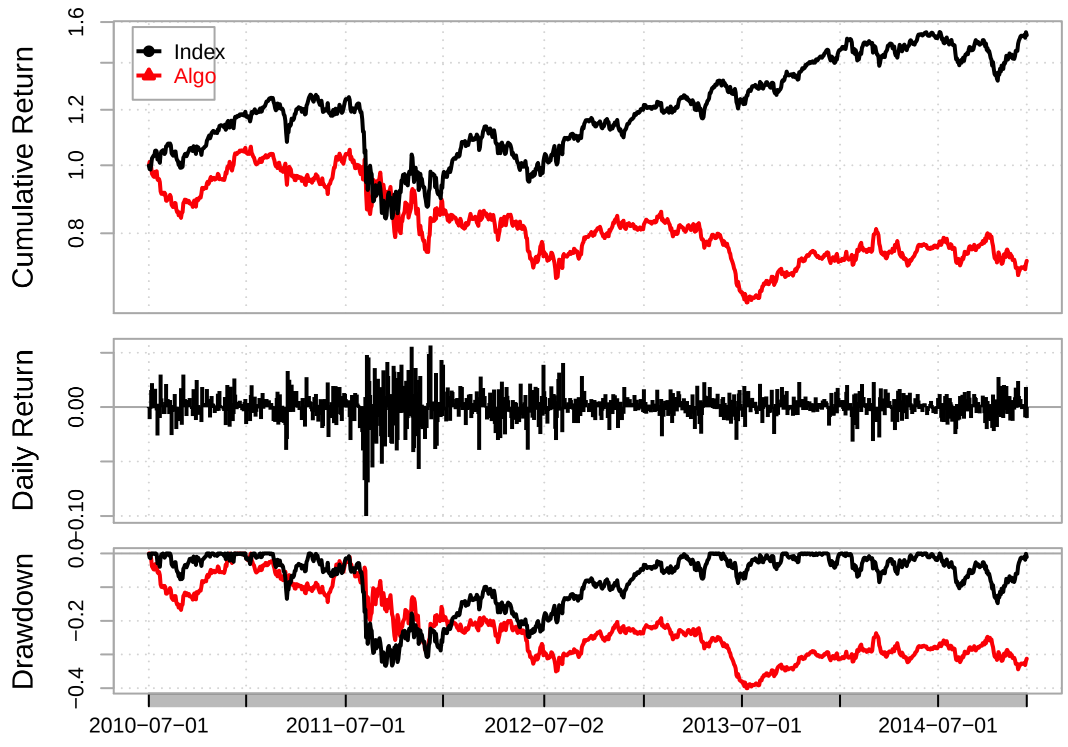


Figure 18: Performance Benchmark Charts for: Standard SVM, 1600 window length, 160 OOS, DAX 30 daily data

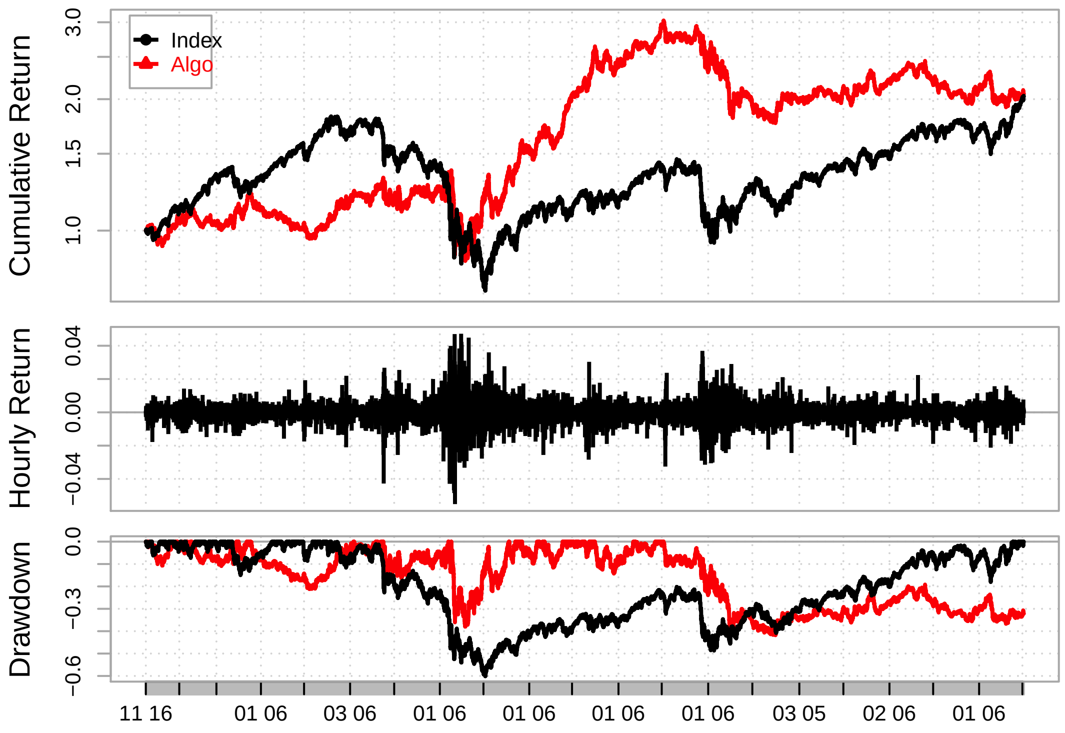


Figure 19: Performance Benchmark Charts for: Standard SVM, 3200 window length, 320 OOS, DAX 30 hourly data

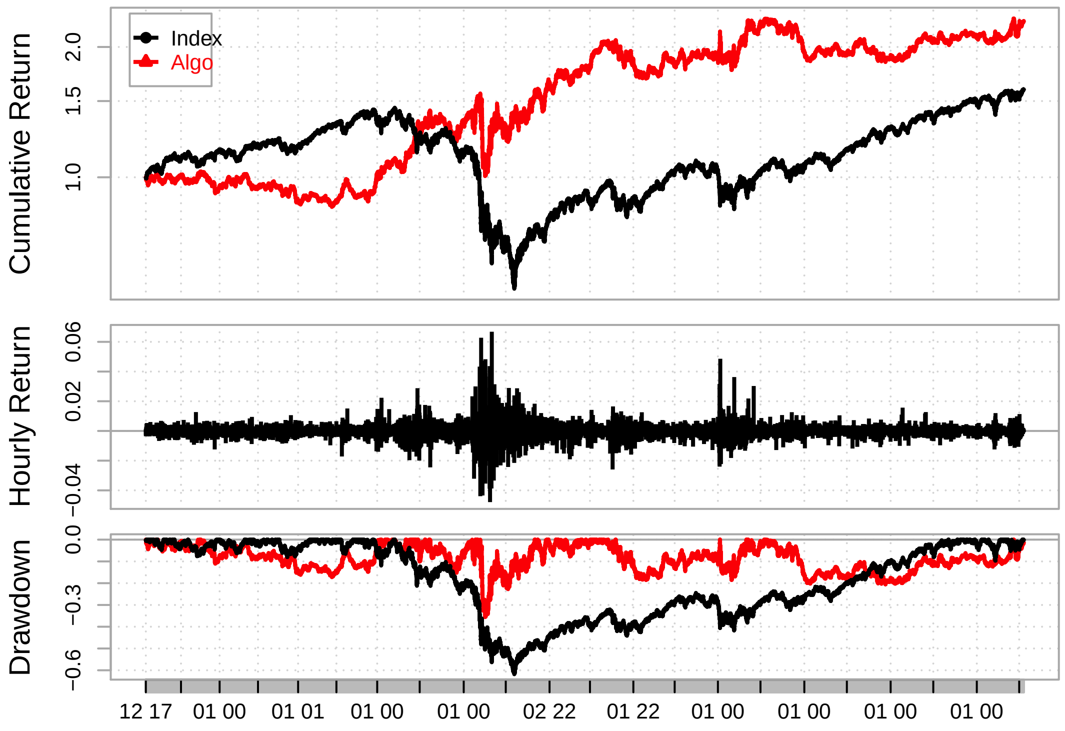


Figure 20: Performance Benchmark Charts for: Standard SVM, 3200 window length, 320 OOS, S&P 500 hourly data

References

[1] Francis E.H Tay and Lijuan Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309 – 317, 2001.

[2] E.E. Peters. *Chaos and Order in the Capital Markets: A New View of Cycles, Prices, and Market Volatility.* Number Bd. 1 in Chaos and Order in the Capital Markets: A New View of Cycles, Prices, and Market Volatility. Wiley, 1996.

[3] Gili Yen and Cheng-few Lee. Efficient market hypothesis (emh): Past, present and future. *Review of Pacific Basin Financial Markets and Policies*, 11(02):305–329, 2008.

[4] Burton G. Malkiel. The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, 17(1):59–82, 2003. available at http://ideas.repec.org/a/aea/jecper/v17y2003i1p59-82.html.

[5] Allan Timmermann and Clive W. J. Granger. Efficient market hypothesis and forecasting. *International Journal of Forecasting*, 20(1):15–27, 00 2004.

[6] Vincenzo Pacelli, Vitoantonio Bevilacqua, Michele Azzollini, et al. An artificial neural network model to forecast exchange rates. *Journal of Intelligent Learning Systems and Applications*, 3(02):57, 2011.

[7] Lamartine Almeida Teixeira and Adriano Lorena Inacio De Oliveira. A method for automatic stock trading combining technical analysis and nearest neighbor classification. *Expert systems with applications*, 37(10):6885–6890, 2010.

[8] Bjoern Krollner, Bruce J. Vanstone, and Gavin R. Finnie. Financial time series forecasting with machine learning techniques: a survey. In *ESANN*, 2010.

[9] Stephen Marsland. *Machine Learning: An Algorithmic Perspective, Second Edition.* Chapman & Hall/CRC, 2nd edition, 2014.

[10] J. Bell. *Machine Learning: Hands-On for Developers and Technical Professionals.* Wiley, 2014.

[11] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[12] Simon Haykin. *Neural Networks and Learning Machines (3rd Edition)*. Prentice Hall, 3 edition, November 2008.

[13] Amparo Albalate and Wolfgang Minker. *Semi-Supervised and Unsupervised Machine Learning*. ISTE/Wiley, London (United Kingdom), 2011.

[14] G.E. Hinton and T.J. Sejnowski. *Unsupervised Learning: Foundations of Neural Computation*. A Bradford Book. MCGRAW HILL BOOK Company, 1999.

[15] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[16] Ruhul A. Sarker and Tapabrata Ray. *Agent-Based Evolutionary Search*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[17] Ruhul A. Sarker, Joarder Kamruzzaman, and Charles S. Newton. Evolutionary optimization (evopt): A brief review and analysis. *International Journal of Computational Intelligence and Applications*, 3(4):311–330, 2003.

[18] Chang Wook Ahn. *Advances in Evolutionary Algorithms: Theory, Design and Practice (Studies in Computational Intelligence)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[19] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.

[20] V. Vapnik and A. Lerner. Pattern Recognition using Generalized Portrait Method. *Automation and Remote Control*, 24, 1963.

[21] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors,

Advances in Neural Information Processing Systems, volume 5, pages 147–155. Morgan Kaufmann, San Mateo, CA, 1993.

[22] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[23] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998.

[24] Ingo Steinwart and Andreas Christmann. *Support Vector Machines.* Springer Publishing Company, Incorporated, 1st edition, 2008.

[25] Shigeo Abe. *Support Vector Machines for Pattern Classification (Advances in Pattern Recognition).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[26] Jorge Nocedal and Stephen J. Wright. *Numerical optimization.* Springer series in operations research and financial engineering. Springer, New York, NY, 2. ed. edition, 2006.

[27] M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control,*, number 25, pages 821–837, 1964.

[28] Thomas M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *Electronic Computers, IEEE Transactions on*, EC-14(3):326–334, 1965.

[29] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004.

[30] Andries P. Engelbrecht. *Computational Intelligence: An Introduction.* Wiley Publishing, 2nd edition, 2007.

[31] Wolfram Schiffmann, Merten Joost, and Randolf Werner. Comparison of optimized backpropagation algorithms. In *ESANN*, volume 93, pages 97–104. Citeseer, 1993.

[32] Brian S. Everitt, Sabine Landau, and Morven Leese. *Cluster Analysis*. Wiley Publishing, 4th edition, 2009.

[33] Peter Hall, Byeong U. Park, and Richard J. Samworth. Choice of neighbor order in nearest-neighbor classification. *Ann. Statist.*, 36(5):2135–2152, 10 2008.

[34] I.T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, 2002.

[35] Lior Rokach. Ensemble-based classifiers. *Artif. Intell. Rev.*, 33(1-2):1–39, February 2010.

[36] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, August 1996.

[37] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[38] Wei Huang, Kin Keung Lai, Yoshiteru Nakamori, and Shouyang Wang. Forecasting foreign exchange rates with artificial neural networks: a review. *International Journal of Information Technology & Decision Making*, 3(01):145–165, 2004.

[39] Erkam Guresen, Gulgun Kayakutlu, and Tugrul U Daim. Using artificial neural network models in stock market index prediction. *Expert Systems with Applications*, 38(8):10389–10397, 2011.

[40] Li-Juan Cao and Francis EH Tay. Support vector machine with adaptive parameters in financial time series forecasting. *Neural Networks, IEEE Transactions on*, 14(6):1506–1518, 2003.

[41] Kyoung jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319, 2003.

[42] R.C. Brasileiro, V.L.F. Souza, B.J.T. Fernandes, and A.L.I. Oliveira. Automatic method for stock trading combining technical analysis and the artificial bee

colony algorithm. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1810–1817, June 2013.

[43] LJ Cao, KS Chua, WK Chong, HP Lee, and QM Gu. A comparison of pca, kpca and ica for dimensionality reduction in support vector machine. *Neurocomputing*, 55(1):321–336, 2003.

[44] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1 – 8, 2011.

[45] Tushar Rao and Saket Srivastava. Modeling movements in oil, gold, forex and market indices using search volume index and twitter sentiments. In *Proceedings of the 5th Annual ACM Web Science Conference*, WebSci '13, pages 336–345, New York, NY, USA, 2013. ACM.

[46] M. Thomason. The practitioner methdods and tool. *Journal of Computational Intelligence in Finance*, 7(3):35–45, 1999.

[47] Iebeling Kaastra and Milton Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215 – 236, 1996. Financial Applications, Part {II}.

[48] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[49] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.

[50] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, pages 507–523, Berlin, Heidelberg, 2011. Springer-Verlag.

[51] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Mach. Learn.*, 46(1-3):131–159, March 2002.

[52] C.R. Bacon. *Practical Portfolio Performance Measurement and Attribution*. The Wiley Finance Series. Wiley, 2005.

Affirmation

I hereby declare that I have composed my Master's thesis "Application and Comparison of different Machine-Learning methods for Financial Time Series Forecasting at the Examples of the DAX 30 and the S&P 500" independently using only those resources mentioned, and that I have as such identified all passages which I have taken from publications verbatim or in substance. Neither this thesis, nor any extract of it, has been previously submitted to an examining authority, in this or a similar form. I have ensured that the written version of this thesis is identical to the version saved on the enclosed storage medium.

Date, signature