Fischer, Thomas; Krauss, Christopher; Treichel, Alex

Working Paper

Machine learning for time series forecasting - a simulation study

FAU Discussion Papers in Economics, No. 02/2018

Provided in Cooperation with:
Friedrich-Alexander University Erlangen-Nuremberg, Institute for Economics

Discussion Papers in Economics

No. 02/2018

Machine learning for time series forecasting - a simulation study

Thomas Fischer
University of Erlangen-Nürnberg

Christopher Krauss
University of Erlangen-Nürnberg

Alex Treichel
University of Erlangen-Nürnberg

ISSN 1867-6707

Friedrich-Alexander-Universität Erlangen-Nürnberg
Institute for Economics
https://www.iwf.rw.fau.de/research/iwf-discussion-paper-series/

Machine learning for time series forecasting - a simulation study

Thomas Fischer[a,1], Christopher Krauss[a,1], Alex Treichel[a,1]

[a] *University of Erlangen-Nürnberg, Department of Statistics and Econometrics, Lange Gasse 20, 90403 Nürnberg, Germany*

Wednesday 24th January, 2018

Abstract

We present a comprehensive simulation study to assess and compare the performance of popular machine learning algorithms for time series prediction tasks. Specifically, we consider the following algorithms: multilayer perceptron (MLP), logistic regression, naïve Bayes, k-nearest neighbors, decision trees, random forests, and gradient-boosting trees. These models are applied to time series from eight data generating processes (DGPs) – reflecting different linear and nonlinear dependencies (base case). Additional complexity is introduced by adding discontinuities and varying degrees of noise. Our findings reveal that advanced machine learning models are capable of approximating the optimal forecast very closely in the base case, with nonlinear models in the lead across all DGPs – particularly the MLP. By contrast, logistic regression is remarkably robust in the presence of noise, thus yielding the most favorable accuracy metrics on raw data, prior to preprocessing. When introducing adequate preprocessing techniques, such as first differencing and local outlier factor, the picture is reversed, and the MLP as well as other nonlinear techniques once again become the modeling techniques of choice.

Introduction

Time series forecasting has been subject to research for many years. By adding a disturbance term to harmonic time series models Yule (1927) effectively introduces the notion that a time series is a realization of a stochastic process. Based on this idea, many concepts have emerged ever since. With the seminal work of Box and Jenkins (1970), research

Email addresses: `thomas.g.fischer@fau.de` (Thomas Fischer), `christopher.krauss@fau.de` (Christopher Krauss), `alex.treichel@fau.de` (Alex Treichel)

in time series analysis has gained further momentum - driven by the introduction of the autoregressive integrated moving average (ARIMA) model and the Box-Jenkins methodology, an approach for "identification, estimation, and verification" of time series (Gooijer and Hyndman, 2006, p. 447). ARIMA models are quite flexible but effectively limited by the assumption of linear intertemporal dependencies. Therefore, they are not able to efficiently capture nonlinear stylized facts, which are often present in real world applications (Zhang et al., 1998, 2001; Gooijer and Hyndman, 2006). Many attempts have been made in that respect - leading to a variety of nonlinear time series models, e.g., bilinear model (Granger and Anderson, 1978), threshold autoregressive (TAR) model (Tong, 2012; Tong and Lim, 1980), smooth transition autoregressive (STAR) model (Chan and Tong, 1986), autoregressive conditional heteroscedastic (ARCH) model (Engle, 1982), generalized autoregressive conditional heteroscedastic (GARCH) model (Bollerslev, 1986; Taylor, 1987), or jump processes (Cox and Ross, 1976), among others. Unfortunately, the majority of these models work well for specific problems, but they do not exhibit a good ability to generalize to other nonlinear modeling tasks.

This insufficiency and the growth of computing power has led to the rise of machine learning models which, compared to classic (nonlinear) time series analysis, do not require prior assumptions about the underlying structure of the data (Zhang et al., 2001; Zhang, 2003). Nonsurprisingly, academics have devoted tremendous work to improving these models and to comparing their forecasting performance with traditional time series methods on artificial and on real world data sets. Today, there exist several interesting areas of research. The first strand of literature focuses on artificial neural networks (ANN) and their relative advantage compared to traditional time series models - see, for example, Chakraborty et al. (1992); Callen et al. (1996); Hill et al. (1996); Kaastraa and Boyd (1996); Zhang et al. (1998); Alon et al. (2001); Zhang et al. (2001); Zhang and Qi (2005). Building upon that, a second strand is devoted to "hybrid methods", i.e., the combination of machine learning with traditional time series approaches such as ARIMA models - see Zhang (2003); Xiao et al. (2012). A third strand focuses on a great variety of (mostly non-ANN) machine learning models in a time series context - see, for example, Kim (2003); Krollner et al. (2010); Ahmed et al. (2010); Bontempi et al. (2013). These three strands of literature share one commonality

- addressing time series prediction tasks with machine learning is gaining momentum in the academic community. Significant innovation occurs in this domain - a few distinct examples follow: Robinson and Hartemink (2010) design a new class of Bayesian networks allowing to cope with nonstationary data generating processes. Durante et al. (2014) develop a locally adaptive factor process that incorporates locally varying smoothness in the mean as well as in the covariance matrix. Finally, Khaleghi et al. (2016) formulate an effective clustering algorithm that assigns two time series to the same class if they were generated from the same distribution. The bottom line is - machine learning research produces powerful and innovative tools for time series analysis tasks.

With our study, we aim at further exploring this exciting field of research at the intersection of machine learning and time series. Specifically, we target the following question: How well do widely used machine learning models fare in forecasting a diverse set of linear and nonlinear time series - in a baseline setting, and when complexity is further increased by introducing jumps or additional noise. Specifically, the contribution of this paper can be divided into three parts.

First, we propose a Monte Carlo study design that allows for comparing the forecasting quality of eight machine learning models (two MLPs, logistic regression, naïve Bayes, k-nearest neighbors, decision trees, random forests, gradient-boosting) across eight linear and nonlinear data generating processes (autoregressive model, bilinear model type 1, bilinear model type 2, nonlinear autoregressive model, nonlinear moving average model, signum autoregressive model, smooth transition autoregressive model, threshold autoregressive model). The salient point is that we make use of a binary target. If the time series rises or remains constant, a "1" is assigned, and a "0" otherwise. This binary response allows for classification accuracy as evaluation metric - which is simple, easy to interpret, and comparable across vastly different DGPs.

Second, we empirically assess the forecasting ability of each machine learning algorithm for each DGP. On a similar note, we derive insights about the "forecastability" of each DGP - by benchmarking the machine learning models against the optimal forecast that can be achieved upon knowledge of the actual process equation. Furthermore, we evaluate the robustness of these findings in light of jumps and varying degrees of noise. We find that the

MLP produces the best results across all considered DGPs in absence of disturbances. By contrast, logistic regression yields the best results in light of noise and high jumps.

Third, we evaluate the effect of different feature engineering and preprocessing techniques to answer the question, whether disturbances can be effectively mitigated and hence learning facilitated. Most notably, we find that outlier detection procedures can significantly enhance the overall accuracy in light of jumps. Also, adding first differences as additional features has a positive net effect.

The remainder of this paper is organized as follows: Section 2 describes the study design, data generation, preprocessing, model training, and evaluation. Section 3 covers the results obtained in the experiments. Section 4 concludes and summarizes the key findings.

Methodology

The methodology covers the entire content and process for setting up our Monte Carlo experiments. First, we discuss how we generate time series data from eight different linear and nonlinear DGPs - including the potential introduction of disturbances via jumps or noise (2.1). Second, we show how we derive features and targets from these time series data, and we introduce different preprocessing techniques, i.e., standardization and outlier removal (2.2). Third, we briefly elaborate on the eight machine learning algorithms we apply, and we discuss how models are trained and how performance is measured (2.3). Fourth, for each DGP, we derive the optimal forecast as ambition level, i.e., the maximum forecasting accuracy that can possibly be attained by any learner (2.4). Finally, we present the parameter configurations for the 29 Monte Carlo experiments we conduct to answer the research questions raised in the introduction (2.5).

Data generation

We use eight linear and nonlinear stochastic processes with normally distributed innovations from which we generate data (2.1.1). To introduce further complexity, we can superpose each DGP either with a compound Poisson jump process (2.1.2) to trigger sudden regime shifts, or with a random walk to inject noise (2.1.3). These DGPs - in combination with controlled distortions, allow for simulating time series data with a varying amount of structure that can be made arbitrarily difficult to capture for the machine learning algorithms. From each of these DGPs, we simulate time series of length $N = 2500$, whereof the first 2000 observations are used for training and the last 500 for evaluating the performance of the machine learning algorithms out-of-sample.

Eight linear and nonlinear data generating processes

We choose eight DGPs - an autoregressive model (AR), a sign autoregressive model (SAR), a nonlinear autoregressive model (NAR), a nonlinear moving average model (NMA), a bilinear model of type 1 (BL1), a bilinear model of type 2 (BL2), a smooth transition autoregressive model (STAR), and a threshold autoregressive model (TAR). There are three motivations for this selection. First, it is based on the study of Zhang et al. (2001) - one of the most comprehensive works in this field of research. Second, these DGPs exhibit varying levels of nonlinearities, ranging from virtually none (AR) to very strong nonlinear dependencies between present and past values (SAR). Third, these DGPs, while by no means comprehensive, still compose a representative subset in contemporary time series literature - for further discussions, see Terasvirta et al. (2010); Terasvirta and Anderson (1992); Granger and Anderson (1978); Tong and Lim (1980); Tong (2012); Tong et al. (1995). We now briefly introduce each DGP. All error terms ε_t are i.i.d. $\mathcal{N}(0,1)$, unless otherwise specified.

Autoregressive (AR) model

$$AR(2): \quad s_t = 0.5 s_{t-1} + 0.45 s_{t-2} + \varepsilon_t. \tag{1}$$

Sign autoregressive (SAR) model

$$SAR(2): \quad s_t = \text{sign}(s_{t-1} + s_{t-2}) + \varepsilon_t, \tag{2}$$

where

$$\text{sign}(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x = 0 \\ -1 & \text{if} \quad x < 0. \end{cases}$$

Nonlinear autoregressive (NAR) model

$$NAR(2): \quad s_t = \frac{0.7|s_{t-1}|}{|s_{t-1}|+2} + \frac{0.35|s_{t-2}|}{|s_{t-2}|+2} + \varepsilon_t. \tag{3}$$

Nonlinear moving average (NMA) model

$$NMA: \quad s_t = \varepsilon_t - 0.3\varepsilon_{t-1} + 0.2\varepsilon_{t-2} + 0.4\varepsilon_{t-1}\varepsilon_{t-2} - 0.25\varepsilon_{t-2}^2. \tag{4}$$

Bilinear (BL) model

$$BL(1): \quad s_t = 0.7s_{t-1}\varepsilon_{t-2} + \varepsilon_t, \tag{5}$$

$$BL(2): \quad s_t = 0.4s_{t-1} - 0.3s_{t-2} + 0.5s_{t-1}\varepsilon_{t-1} + \varepsilon_t. \tag{6}$$

Smooth transition autoregressive (STAR) model

$$STAR(2): \quad s_t = 0.3s_{t-1} + 0.6s_{t-2} + (0.1 - 0.9s_{t-1} + 0.8s_{t-2})[1 + \exp(-10s_{t-1})]^{-1} + \varepsilon_t. \tag{7}$$

Threshold autoregressive (TAR) model

$$TAR(2): \quad s_t = \begin{cases} 0.9s_{t-1} + 0.05s_{t-2} + \varepsilon_t & \text{for} \quad |s_{t-1}| \leq 1 \\ -0.3s_{t-1} + 0.65s_{t-2} - \varepsilon_t & \text{for} \quad |s_{t-1}| > 1. \end{cases} \tag{8}$$

Jump process

Jumps represent a form of discontinuity in the series. Depending on number of occurrence and size, they can have a substantial impact on the machine learning models' performance, e.g., by provoking a mean shift. In some Monte Carlo settings, we pollute the original DGP s_t with jumps by superposing s_t with a compound Poisson process j_t as follows,

$$x_t = s_t + j_t, \tag{9}$$

where x_t denotes the resulting DGP. The compound Poisson jump process j_t is defined as in Ross (2007) by

$$j_t = \sum_{i=1}^{N_t} D_i \quad \text{with} \quad D_i \sim \mathcal{N}(0, s_p^2) \quad \text{and} \quad N_t \sim \mathcal{P}(\lambda_p), \tag{10}$$

where s_p controls the jump size and λ_p the jump rate, at which the jumps are expected to occur. For the jump experiments we set s_p to 0.1, 1, and 10, i.e., 10 %, 100 %, or 1000 % of the standard deviation of a regular innovation to reflect increasingly higher levels of disturbance. The larger s_p, the larger the variance of the distribution from which the jumps are drawn, and the larger is the average jump in magnitude (while the mean over positive and negative jumps is still zero). The parameter λ_p is set to $N/10$, where N denotes the length of the generated time series. In other words, a jump is supposed to occur after the passing of λ_p periods in expectation. Thus, for $N = 2500$, 10 jump events are expected to happen, corresponding to 0.4% of all observations. This parameter is selected in accordance with the jump analysis of Jondeau et al. (2015). Superposing the DGP s_t with the compound Poisson Jump process results in a corresponding mean shift by the actual jump size that occurred at each jump event. With the jump size parameter s_p, we can hence adjust the expected level of distortion occurring at each event.

Noise process

In some Monte Carlo settings, we pollute the original DGP s_t with a stochastic mean drift by superposing s_t with a random walk n_t as follows,

$$x_t = s_t + n_t, \tag{11}$$

where x_t denotes the resulting DGP, and where n_t is given by

$$n_t = n_{t-1} + \varepsilon_t \quad \text{with} \quad \varepsilon_t \sim \mathcal{N}(0, \sigma_N^2). \tag{12}$$

As is customary in the fields of electrical and telecommunications engineering, the realization of different levels of noise leads to our definition of the *signal to noise ratio*, SNR. We compute it as the ratio of the unconditional variance of the structured process σ_S^2 and the variance of the innovations of the random walk[1] σ_N^2,

$$SNR = \frac{\sigma_S^2}{\sigma_N^2}, \tag{13}$$

where σ_S^2 is empirically derived.

For this simulation, we choose four different noise levels:

- No noise: $SNR \to \infty$
- Low noise: $SNR = 20$
- Medium noise: $SNR = 4$
- High noise: $SNR = 2$

With the superposition in equation (11), we achieve a resulting DGP x_t that is globally nonstationary due to the random walk overlay. However, depending on SNR, the process may still be locally predictable - as long as the stochastic mean drift remains small compared to the deterministic components in the original DGP s_t. By decreasing SNR, we can hence increase the level of pollution by stochastic mean drift - and assess the reaction of different machine learning algorithms.

Prediction target, feature engineering and preprocessing

Prediction target

As outlined in the beginning, we use a binary prediction target to ensure comparability of the results across DGPs - even irrespective of the introduced distortions, i.e., jumps and noise. In particular, we compute a binary target y_t specifying whether the realization of the process $(x_t)_{t\in T}$ at time t is larger than or equal to its value in $t-1$ or not[2]

$$y_t = \begin{cases} 1 & \text{if} \quad x_t - x_{t-1} \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

[1]We use the variance of the innovations of the random walk, given that its unconditional variance grows linearly with t.

[2]Please note that unless otherwise stated, x_t always refers to the process regardless of whether or not is is superposed with noise or jumps. Specifically, x_t equals s_t in case no distortions are present.

Feature engineering

Feature engineering is a crucial element in data science. In our context, features are generally derived from historical time series data prior to or at the time of prediction. We use a variety of features in different combinations to examine their impact on the results (Ahmed et al., 2010).

- Lagged values $x_{t-1}, x_{t-2}, ..., x_{t-N}$ (LAGS): In the default case, we take the past two lagged time series values as features. Furthermore, we vary the number of lagged values between one and five to analyze their impact on the quality of the predictions. Regarding our DGPs outlined in subsection 2.1.1, two lags would be optimal, but we aim to assess the robustness versus using fewer or more lags (model mis-specification).
- First differences $\Delta x_{t-1,t-2}, \Delta x_{t-2,t-3}, ..., \Delta x_{t-N-1,t-N}$ (DIFF): The first backward difference of the time series. We expect this feature to improve classification accuracy under the influence of nonstationarity, given that first differences are independent of the level of the time series. When using first differences as features, we always feed the last two lagged values, i.e., $\Delta x_{t-1,t-2}, \Delta x_{t-2,t-3}$.
- Moving averages (MA): Smoothed representations of the time series by simple moving averages with window sizes W of 2, 4, and 8:

$$z_t^{(W)} = \frac{1}{W} \sum_{i=1}^{W} x_{t-W+i} \tag{15}$$

  where $z_t^{(W)}$ denotes the time series value at time t smoothed over a window of size W. We hypothesize that these features may improve classification accuracy by smoothing out noise (Ahmed et al., 2010). When adding MA, we use the first lagged value of each moving average as input feature, i.e., $z_{t-1}^{(W)}$.

Preprocessing

We perform two kinds of preprocessing, standardization and outlier removal.

Standardization: Standardizing means bringing each column in a feature matrix to the same scale. Features of varying scale are hurtful to the learning process of many algorithms. For example, distance-based algorithms typically malfunction, if features contribute unequally to the total distance (Qian et al., 2004). But standardization can also have a pos-

Figure 1: Procedure to remove jumps with LOF.

itive effect on training times. For example, algorithms relying on stochastic gradient descent (SGD) typically converge faster when training on normalized data (Ioffe and Szegedy, 2015). We standardize by subtracting the column mean from each column in the feature matrix and dividing by the column standard deviation both obtained from the training set, i.e., the features corresponding to the first 2000 observations of the time series.

Outlier removal: The outlier removal procedure is based on the well known anomaly detection algorithm "local outlier factor" (LOF) by Breunig et al. (2000). For each data point, a local density is estimated based on its k nearest neighbors and a score is computed with that density. If a data point's outlier factor significantly deviates from those of its neighbors, it is considered an outlier. This procedure hence requires multiple steps and is depicted in figure 1.

First, we start by identifying the outliers and their positions in the time series with the LOF algorithm by applying it to the first differences of the time series. Second, we assume that each of these outliers results in a mean shift, and we construct a time series of successive mean shifts - a compound Poisson process. Third, we subtract the estimated compound Poisson process from the raw time series. This procedure can be understood as the empirical inverse to (9), whereby jump position and sizes are estimated from the data in order to eliminate the mean shifts. Finally, for the residual series, we compute the mean, section-by-section between jumps, and subtract it from the residual series - hence eliminating all remaining mean shifts. To avoid look-ahead bias, we perform this procedure in an expanding window approach. Specifically, we apply these steps on the whole training set, then on the whole training set and the first out of sample observation, then on the whole training set and the first two out of sample observations, and so forth.

Model training and evaluation

Overview of applied models

It is impossible to consider the abundance of machine learning models existing today. Therefore, we only concentrate on the most common algorithms which are also accessible through the Python package scikit-learn (Pedregosa et al., 2011).

Model parametrization and validation are very challenging tasks. Ideally, one would specify a range for each parameter, iterate through it in a grid search manner and validate the models to identify the optimal parameter constellation from the grid. This approach is extremely time consuming and exceeds the scope of our simulation study by far. Just to clarify - we would have to conduct such parameter tunings one time for each model and DGP - resulting in a total of 64 grid searches for each Monte Carlo setting. Instead, we parameterize our models with specifications that are supposed to work across a wider range of applications, following recommendations from the literature.

Multilayer perceptron (MLP)

The MLP is a very popular form of artificial neural networks that can be applied to both regression and classification problems (Zhang et al., 1998; Zhang, 2003; Hastie et al., 2008). MLPs are fully connected multilayer feedforward networks composed of several layers. Hereby, the first layer represents the input of the network and the last layer provides its output. Between these layers, one or more so called, hidden layers are arranged. We apply the most widely used form of MLPs which consists of exactly one hidden layer.

Mathematically, MLPs can be interpreted as nonlinear function mappings $f_k(X)$ from the inputs to the outputs. For a general K-class classification problem, we can hence describe the network as a series of equations (Hastie et al., 2008):

$$
\begin{aligned}
Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, ..., M, \\
T_k &= \beta_{0k} + \beta_k^T Z, \quad k = 1, ..., K, \\
f_k(X) &= g_k(T), \quad k = 1, ..., K.
\end{aligned} \tag{16}
$$

Thereby, X denotes the input vector of P features, α_{0m}, β_{0k} are bias units, α_m, β_k are the weights connecting the m^{th} hidden neuron with each of the P features, and the k^{th} output neuron with each of the M hidden neurons respectively. Furthermore, $\sigma(\cdot)$ and $g_k(\cdot)$ denote activation functions. The three equations work as follows. First, each of the M hidden neurons receives a linear combination of the P inputs, and nonlinearly transforms it with $\sigma(\cdot)$ to a derived feature Z_m. Second, the M derived features are fed into each of the K output neurons with their respective weight, where they are nonlinearly transformed with $g_k(\cdot)$ to K outputs - corresponding to K classes.

In the literature, there is a very wide range of choices and rules of thumb for the number of hidden layers, neurons, and activation functions. We hence decide to work with two generic MLPs. The first, smaller net contains one hidden layer with $M = 10$ hidden neurons (MLP10); the second, larger net has one hidden layer with $M = 100$ neurons. In case of two input features, this results in 41 parameters that need to be trained for the MLP10 and 401 for the MLP100 respectively[3]. In both networks, we make use of the rectifier linear unit function in the hidden layer - a common, typically well-performing choice and the default in scikit-learn (Pedregosa et al., 2011). For the output layer, we only use one neuron with a sigmoid activation function. Hence, the output $f_1(X)$ can be interpreted as the probability

[3] $2 \times 10 + 10 \times 1$ coefficients and $10 + 1$ bias terms for the MLP10; $2 \times 100 + 100 \times 1$ coefficients and $100 + 1$ bias terms for the MLP100.

that our time series rises or remains constant in period t (corresponding to class "1"). The probability that the time series falls in t (class "0") is hence $1-f_1(X)$.[4] Note that the number of input neurons is given by the number of features for both architectures.

Logistic regression (LR)

Despite its name, it should be emphasized that logistic regression is an algorithm for classification settings. It has been developed to model the posterior probabilities of K possible outcomes (classes) by linearly combining the inputs x. Hereby, the LR model ensures that the probabilities of all possible classes K sum up to 1 while each individual probability has a value between 0 and 1. For a K-class classification problem, the model consists of $K-1$ log odds of the following form (Hastie et al., 2008):

$$\begin{aligned}
\log\frac{P(K=1|X=x)}{P(K=K|X=x)} &= \beta_{10}+\beta_1^T x \\
\log\frac{P(K=2|X=x)}{P(K=K|X=x)} &= \beta_{20}+\beta_2^T x \\
&\dots \\
\log\frac{P(K=K-1|X=x)}{P(K=K|X=x)} &= \beta_{(K-1)0}+\beta_{(K-1)}^T x
\end{aligned} \tag{17}$$

Hereby, $P(K=k|X=x)$ with $k\in\{1,2,...,K-1\}$ is the posterior probability of class k given the input vector of observation x. Furthermore, β_{k0} denotes the intercept and β_k the coefficients for the respective class k. As our classification problem consists of two classes ("upward" and "downward"), the model becomes a single linear function, i.e., $\beta_{10}+\beta_1^T x$ that can be estimated by maximum likelihood using the N observations from the training data:

$$L(\beta)=\sum_{i=1}^{N}\left\{y_i\beta^T x_i-\log(1+\exp(\beta^T x_i))\right\} \tag{18}$$

In the above loss function L, β comprises $\{\beta_{10},\beta_1\}$ and the input vector x_i is extended by the constant "1" to also incorporate the intercept.

For this study we use *LogisticRegressionCV* from the scikit-learn package (Pedregosa et al., 2011) as LR implementation which performs a grid search on the $l2$ regularization

[4] Alternatively, a softmax activation function with $K=2$ output neurons could have been chosen.

parameter in the range from 10^{-4} to 10^{4}. Furthermore, we set *max_iter* to 5000 and use the default algorithm *liblinear* as solver.

Naïve Bayes (NB)

Naïve Bayes classifiers are based on the Bayes' theorem and have proven to be fast, reliable and very effective in many different applications including spam filtering, taxonomic studies, and decision making in medical treatments (Raschka, 2014). The designation "naïve" relates to the NB assumption that the features are mutually independent - an assumption which in general is not true (Raschka, 2014; Hastie et al., 2008). Nonetheless, NB classifiers often outperform more complex machine learning models, even though its assumptions are not fully met (Hastie et al., 2008; Lewis, 1998; Hand and Yu, 2001). The base principle behind NB classifiers is the calculation of posterior probabilities, i.e., the probability $P(K = k|X = x)$ that an observation x belongs to class k given its features. Following Raschka (2014), this can be formalized as:

$$P(K = k|X = x) = \frac{P(X = x|K = k)P(K = k)}{P(X = x)} \tag{19}$$

The predicted class label for an observation x is then the class k which has the highest posterior probability according to equation (19).

For our simulation study we use the *GaussianNB* classifier from the scikit-learn package, where the likelihood of the features is assumed to be Gaussian (Pedregosa et al., 2011).

k-Nearest neighbors (kNN)

The kNN classifier is one of the simplest machine learning techniques. The class of a given data point x_0 is found by performing a majority vote among the k neighbors which are closest in distance d_i to x_0 (Bishop, 2007). A number of distance metrics are possible but the most common one is the Euclidean distance, given as.

$$d_i = ||x_i - x_0||. \tag{20}$$

In our experiments, we take 5 neighbors into account, reflecting the default value in the scikit-learn package - often a good compromise to limit overfitting (which occurs for small k) while still obtaining meaningful decision boundaries (Pedregosa et al., 2011).

Classification and regression trees (CART)

As suggested by the name "classification and regression trees", CARTs can be applied to both classification and regression problems. They operate by learning a series of decision rules from the training data, based on which they infer the value (regression) or class label (classification) of new observations (Raschka, 2015). Hereby, the decision rules are computed in a way that they separate the samples of each class as effectively as possible, i.e., by maximizing the information gain at each split[5]. This procedure starts at the root of the tree and is recursively repeated at each node until the samples in all child nodes of the tree are pure or another stop criterion, e.g., maximum tree depth is reached. Following Raschka (2015), the information gain IG for a binary split can be formalized as

$$IG(D_{parent}, f) = I(D_{parent}) - \left[\frac{N_{left\_child}}{N_{parent}} I(D_{left\_child}) + \frac{N_{right\_child}}{N_{parent}} I(D_{right\_child}) \right]. \quad (21)$$

Thereby, D_{parent}, D_{left_child} and D_{right_child} denote the data sets prior to and after the split with feature f, N_{parent}, N_{left_child} and N_{right_child} denote the corresponding number of samples, and $I(\cdot)$ denotes the applied impurity measure. The information gain IG is then defined as the difference in impurity prior to $(I(D_{parent}))$ and after the split (weighted impurities $I(D_{left\_child})$ and $I(D_{left\_child})$).

For our study, we make use of the DecisionTreeClassifier implementation from the scikit-learn package with the gini impurity measure (scikit-learn default) and a maximum tree depth of 10 (Pedregosa et al., 2011).

Random forests (RF)

Random Forests (Breiman, 2001; Ho, 1995) are an ensemble technique averaging the predictions of a large number of decorrelated decision trees. They usually show good performance with better generalization properties than individual trees, are relatively robust to outliers and require virtually no parameter tuning (Raschka, 2015). Random forests are built on two main ideas - bagging to build each tree on a different bootstrap sample of the training data,

[5] A potential decision rule based on the feature "value of the time series as lag 1" could be whether or not the value is larger than 0.5.

and random feature selection to decorrelate the trees. The training algorithm is fairly simple and can be summarized along the lines of Hastie et al. (2008): For each of the B trees in the ensemble, draw a bootstrap sample Z from the training data. When growing the tree T_b on Z, randomly draw m of the p features which are available as candidates for the split in the respective node. Finally, add the grown tree T_b to the ensemble. During inference, each of the B trees makes a prediction $\hat{c}_b(x)$ for the class label of the new observation x. The final prediction of the random forest $\hat{c}_{RF}(x)$ is then the majority vote of the B trees, i.e., $\hat{c}_{RF}(x) = majority\ vote\{\hat{c}_b(x)\}_1^B$.

Inspired by Breiman (2001) we proceed with a RF containing 100 trees, each with a max depth of 10 for our simulation study. All our trees use cross-entropy as impurity measure and we set $m = \sqrt{p}$ features - the default choice for classification settings.

Gradient boosting trees (GBT)

Boosting (Schapire, 1990) is another ensemble technique, which additively combines many weak learners to a powerful ensemble. As opposed to random forests, where each tree is independently built on a bootstrap sample, GBT grows a sequence of trees $T_m(x)$, $m = 1, 2, 3, \ldots$ on repeatedly adjusted versions of the training data (Hastie et al., 2008). Specifically, after each boosting iteration (one iteration is the fitting of one tree), the weights of the previously misclassified observations are increased and the weights of the correctly classified samples are decreased. As a result, all successive trees in the sequence put more attention on the "difficult" observations. During inference, and as in the random forest case, each of the M trees make its prediction $\hat{f}_m(x)$. The final prediction of the GBT $\hat{f}_{GBT}(x)$ is then a weighted majority vote:

$$\hat{f}_{GBT}(x) = sign\big(\sum_{m=1}^{M} w_m \hat{f}_m(x)\big). \tag{22}$$

Thereby, w_m, $m = 1, 2, 3, \ldots$ denotes the weight of the prediction of the respective tree $\hat{f}_m(x)$ which has been computed by the boosting algorithm during the training phase.

For our simulation study, we follow Friedman (2002) and employ a GBT consisting of 100 shallow decision trees with a maximum depth of 3. Furthermore, we set the learning rate to 0.1 - the default value in the scikit-learn software package (Pedregosa et al., 2011).

Comparison metric and model performance evaluation

Even though recent research focuses of cross-validation techniques (CV) (Kohavi et al., 1995; Bergmeir and Benítez, 2012; Bergmeir et al., 2014, 2015), we apply a classic out-of-sample validation scheme. The latter is still highly accepted, commonly used, less computationally expensive, and preserves the time structure of our data. To be precise, we perform an 80/20 split. Therefore, the first 2000 observations of the time series belong to the training set and the last 500 observations to the holdout test set, on which the results are evaluated. As evaluation metric, we opt for classification accuracy. The proportion of correctly classified samples to all samples is very easy to interpret and allows us to compare between DGPs with different structure. Also, this metric is numerically stable and works well for all models.

Obtaining the benchmark - ambition level analysis

Prior to evaluating the different machine learning models against each other, we take one intermediate step to establish a benchmark for each DGP. Specifically, we compute the optimal forecast, using the model equation of the respective DGP. The predictive accuracy of the optimal forecast constitutes an "ambition level", i.e., the best result we could possibly attain with a machine learning algorithm. Note that a related exercise has also been conducted by, e.g., Connor et al. (1994) in a different setting.

To derive the ambition levels, we proceed as follows. For each DGP, we use the respective process equation as outlined in subsection 2.1.1 to compute the optimal forecast $\hat{s}_t$ as

$$\hat{s}_t = E(s_t|\mathcal{F}_{t-1}) = E(s_t|s_{t-1}, s_{t-2}, \ldots), \tag{23}$$

where $\mathcal{F}$ denotes the information set up until time $t-1$. When computing expectations, we set ε_t to its expected value of zero and all other lagged terms to their true values. Then, we transform this forecast value into a binary prediction by assigning a "1" in case $\hat{s}_t$ is larger or equal to s_{t-1}, and a "0" otherwise. Finally, we compute the achieved accuracy by comparing the prediction with the actual binary target[6]. To increase robustness of the results, we perform 1000 replications in a Monte Carlo setting, and average the results per DGP.

[6]Again, a "1" is assigned in case s_t is larger than or equal to s_{t-1}, and a "0" otherwise.

Table 1 shows the ambition level for each DGP. We make two main observations. First, the analysis clearly underlines the stochastic nature of the DGPs. Even though we have perfect knowledge of the process equations, the highest predictive accuracy is 80.71% (NMA). Second, the analysis reveals strong differences in terms of "forecastability" between the DGPs, ranging from an accuracy of 65.55% for AR(2) up to 80.71% for NMA[7]. The differences can be attributed to the relative importance of the error term ε_t compared to the deterministic part. The intuition is as follows: The stronger the deterministic movement ($|\hat{s}_t - s_{t-1}|$), the higher ε_t needs to be in order to change the direction of the overall movement. Such an event is more likely, when the deterministic movement is small.

DGP	AR(2)	BL(1)	BL(2)	NAR(2)	NMA	SAR(2)	STAR(2)	TAR(2)	∅
Accuracy	65.55%	76.40%	73.82%	73.81%	80.71%	71.70%	77.73%	76.21%	74.49%

Table 1: Ambition levels, i.e., best possible accuracy that can be achieved with the optimal forecast for each of the eight DGPs.

Another angle to look at the results is the complexity of the decision boundary, i.e., the hyperplane separating the two classes (Dreiseitl and Ohno-Machado, 2002). Figure 2 shows the decision boundary obtained from the optimal forecast[8]. Each plot can be read as follows: The x-axis represents the value of the process realization at time $t-1$ (i.e., s_{t-1}) and the y-axis the value at $t-2$, (i.e., s_{t-2}). Each contour plot hence spans a grid with all possible combinations of s_{t-1} and s_{t-2} with the shading of the areas denoting the value of the prediction[9]. The lighter shaded areas represent a prediction of "1", i.e., $\hat{s}_t \geq s_{t-1}$, and the darker areas a prediction of "0". For illustrative purposes, we have included the decision boundary for an AR(1) process in the upper left corner. As expected, we make two observations. First, and as the AR(1) process equation suggests, the forecast solely depends on the value of s_{t-1} and the decision boundary hence runs in parallel to the y-axis. Second,

[7]Please note that the findings in table 1 are specific to the concrete coefficients of the process equations outlined in subsection 2.1, i.e., they do not imply that an AR(2) process is in general harder to predict than an NMA process.

[8]An introduction on contour plots to visualize decision boundaries is available in Raschka (2015).

[9]An exception forms the NMA process where the x-axis represents ε_{t-1} and the y-axis ε_{t-2}.

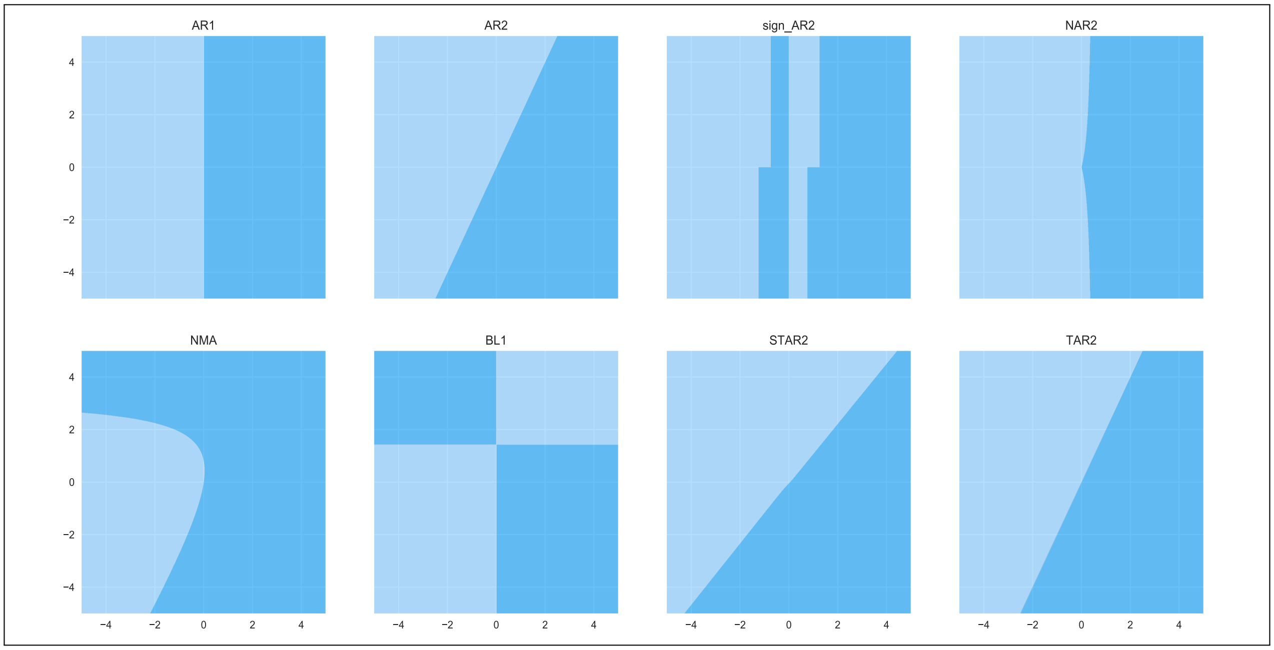

Figure 2: Optimal decision boundaries by DGP. The lighter shaded areas indicate a forecast value of "1", i.e., $\hat{s}_t$ is larger than or equal to s_{t-1}; the darker shaded areas a forecast value of "0". The x-axis denotes s_{t-1}, the y-axis s_{t-2}. An exception forms the NMA process, where the x-axis denotes ε_{t-1} and the y-axis ε_{t-2}. Furthermore, we have omitted the BL2 process, as it cannot be plotted in the same logic in two dimensions.

the forecast is "1" for all values $s_{t-1} \geq 0$ and "0" for all values $s_{t-1} < 0$. This corresponds to the mean reversion property of the AR(1) process. All other plots can be interpreted in a similar manner - even though some of the decision boundaries have a more complex, in some cases clearly nonlinear shape. Surprisingly, DGPs such as STAR2 with its coefficients obtained from the literature (see Zhang et al. (1998)) seem to be almost linearly separable - despite their general ability to describe nonlinear intertemporal dependencies.

Monte Carlo simulations

In total, we conduct 29 Monte Carlo experiments in in four groups - base analysis, lag analysis, jump effect analysis, and noise analysis. Each group aims at answering the research questions raised in the introduction. Details are presented here below and in table 2.

(1) Base analysis: The objective is to predict directional movement of time series generated from the DGPs outlined in subsection 2.1. We use the past two lags as input features,

and do not introduce any additional form of disturbance on top of the stochastic nature of the error terms ε_t. This analysis is conducive in two respects. First, we are able to assess how close each of the machine learning algorithms gets to the ambition level. Second, in subsequent experiments, the findings serve as a base case to analyze the impact of further distortions.

(2) Lag analysis: The second group analyses the impact of the number of lagged values as input features. In the base case, we feed two lagged values - corresponding to the number of lags in the underlying DGPs. The lag analysis reveals the impact of model mis-specification, i.e., when using a higher or lower number of lags ranging from 1 to 5.

(3) Jump effect analysis: We evaluate the impact of permanent jumps of different sizes by superposing compound Poisson jump processes on the original DGPs - see subsection 2.1. Subsequently, we evaluate different variants of feature engineering and preprocessing to balance the adversarial effect of jumps. Specifically, we consider only lags as features (3a), lags and first differences (3b), lags and LOF preprocessing (3c), as well as lags, first differences, and LOF preprocessing (3d). For a review of these techniques, see subsections 2.2.2 and 2.2.3. As outlined in subsection 2.1.2, each DGP is expected to exhibit 10 jumps on average with magnitudes of 0.1, 1, and 10. The number of lags and the degree of noise is the same as in the base case.

(4) Noise analysis: In the fourth group, we evaluate the impact of stochastic mean drifts by superposing a random walk on the original DGPs - see subsection 2.1.3. Next, we assess whether feature engineering may help to eliminate some of the noise. Specifically, we consider only lags as features (4a), lags and first differences (4b), lags and moving averages (4c), as well as lags, first differences, and moving averages (4d). Noise levels are low, medium, and high, following the definitions in 2.1.3.

For each Monte Carlo experiment, we generate 1000 time series of length 2500 from each DGP. The first 2000 observations are used for training, and the last 500 observations for validation. All eight models are individually trained and validated on each of the 1000 time series per DGP. Averaging predictive accuracies over all 1000 time series per DGP per model provides us with a statistically robust assessment of model performance.

		Distortions		Applied feature engineering & preprocessing			
Group	**#**	**Jump size**	**Noise level**	**MA**	**DIFF**	**LAGS**	**LOF**
Base analysis	1		no noise			2	
Lag analysis	2		no noise			1	
	2		no noise			3	
	2		no noise			4	
	2		no noise			5	
Jump analysis	3a	0.1	no noise			2	
	3a	1	no noise			2	
	3a	10	no noise			2	
	3b	0.1	no noise		+	2	
	3b	1	no noise		+	2	
	3b	10	no noise		+	2	
	3c	0.1	no noise			2	+
	3c	1	no noise			2	+
	3c	10	no noise			2	+
	3d	0.1	no noise		+	2	+
	3d	1	no noise		+	2	+
	3d	10	no noise		+	2	+
Noise analysis	4a		low noise			2	
	4a		medium noise			2	
	4a		high noise			2	
	4b		low noise		+	2	
	4b		medium noise		+	2	
	4b		high noise		+	2	
	4c		low noise	+		2	
	4c		medium noise	+		2	
	4c		high noise	+		2	
	4d		low noise	+	+	2	
	4d		medium noise	+	+	2	
	4d		high noise	+	+	2	

Table 2: Overview of conducted Monte Carlo experiments.

Empirical results

In this chapter we finally apply the machine learning models to the outlined prediction task. For illustration of the effects of our mitigation and improvement measures we make use of the relative change to the base case in percent given by $\left(\frac{\textit{New absolute value}}{\textit{Abolute base case value}} - 1\right) \times 100$. Where appropriate, we selectively include plots of the obtained decision boundaries to further illustrate some of the observed effects.

Base analysis

The goal of the base analysis is threefold. First, we aim to determine how close the different machine learning models come to the best possible result obtained during the ambition level analysis (see subsection 2.4). Second, we wish to identify differences in "predictability" between DGPs, i.e., whether certain DGPs are easier or harder to predict across all our applied machine learning models. Third, we aim to spot particularities on the level of DGP and model, e.g., whether specific models should be avoided for certain DGPs.

	Panel A		Panel B							
DGP	**Ambition level**	**ML models ∅**	**LR**	**MLP10**	**MLP100**	**CART**	**RF**	**GBT**	**kNN**	**NB**
AR(2)	65.55%	63.37%	65.47%	65.35%	**65.48%**	60.45%	63.31%	64.15%	60.72%	62.07%
BL(1)	76.40%	72.72%	**74.14%**	73.86%	73.83%	69.66%	72.60%	73.23%	70.79%	73.66%
BL(2)	73.82%	69.64%	68.70%	71.50%	**72.00%**	67.76%	70.56%	71.22%	68.81%	66.55%
NAR(2)	73.81%	72.49%	73.64%	73.63%	**73.69%**	69.80%	72.53%	72.88%	70.24%	73.52%
NMA	**80.71%**	**78.76%**	79.76%	79.91%	**79.97%**	75.85%	78.70%	79.29%	77.21%	79.42%
SAR(2)	71.70%	65.80%	61.18%	67.81%	69.05%	66.63%	69.61%	**70.57%**	66.81%	54.72%
STAR(2)	77.73%	76.03%	77.54%	77.54%	**77.57%**	72.38%	75.96%	76.82%	74.51%	75.95%
TAR(2)	76.21%	75.51%	76.10%	76.04%	75.97%	74.34%	75.12%	75.41%	74.87%	**76.20%**
∅	74.49%	71.79%	72.07%	73.21%	**73.44%**	69.61%	72.30%	72.95%	70.49%	70.26%

Table 3: Panel A and B summarize the achieved accuracies of the machine learning models in the base case. Panel A contrasts the best possible accuracy (ambition level) with the average performance achieved across all our machine learning models. Panel B further details the results for each model and DGP individually and shows the results on model-level, i.e., the average performance of a specific machine learning model across all DGPs (column average at the very bottom). The highest value of the respective column/row is highlighted in bold.

Model performance: Looking at the column average in Panel B of table 3, we see that the MLP100 (73.44%), the MLP10 (73.21%), and the GBT (72.95%) cope best with the broad mix of linear and nonlinear DGPs. All three models come close to the overall ambition level of 74.49%. In particular, the MLP100 achieves first place in five of the eight examined DGPs and is only outperformed once by the GBT in case of the SAR(2) process with a clear margin of 1.52% (69.05% vs. 70.57%). The underlying reason is related to the straight edges and rectangular shape of the decision boundary of the SAR(2) process which is natural to and hence advantageous for the GBT as well as the other tree-based machine learning models (see also deep dive on decision boundaries further below).

With an average accuracy of 72.07%, the LR still presents a good choice in many cases, especially when considering its usually low computational costs. Looking at the details in Panel B, we find the LR to be competitive when applied to most of the DGPs - the exception are BL(2) and SAR(2), where its performance drops significantly. Both of these DGPs are characterized by strongly nonlinear decision boundaries (see figure 2), where simpler model classes reach their limit.

The lower end of the ranking consists of CART (69.61%) followed by NB (70.26%) and kNN (70.49%). All of these models, with the exception of NB for TAR(2), clearly fall behind the top tier. In case of CART, the underperformance can be attributed to overfitting caused by the relative high tree depth of 10. In case of NB, the root cause for the poor performance lies in the autocorrelated nature of the DGPs, leading to correlated features, and thus violating the NB assumption of independence between every pair of features.

Looking at the different DGPs in general (see Panel A of table 3), we see that the DGPs with linear decision boundaries (group 1), e.g., AR(2) can be well predicted with both linear and nonlinear machine learning models. Group 2, i.e., STAR(2) and TAR(2) which both show weak nonlinearities, are still a good fit for linear models such as the LR, as the model learns a fairly good approximation of the decision boundary. Finally in group 3 (SAR(2) and BL(2)), the performance of the linear models drops significantly as the nonlinearities of the DGPs become too strong.

Deep dive on decision boundaries: To further illustrate some of the insights described above, we visualize the decision boundaries learned by the different machine learning models. Figure 3 exemplarily depicts the decision boundaries for the SAR(2) process. The sub plot on the top left corresponds to the decision boundary obtained from the optimal forecast (see subsection 2.4). The other eight sub plots show the decision boundaries learned by the different machine learning models. Each of these plots can be interpreted similar to section 2.4 (ambition level analysis) with the lighter shaded areas denoting the prediction of an "upward" and the darker shaded areas the prediction of a "downward" movement of the process in the next period t. The colored points follow the same color coding and represent a subset of the actual observations that were used to fit the machine learning models. We make the following observations: Looking at the optimal decision boundary on the plot on the top left, we see that the prediction ("1" versus "0") is almost solely dependent on the value of the SAR(2) process at time $t-1$ (x-axis). Two exceptions can be observed around the middle of the plot where the value of the process at time $t-2$ has an impact on the prediction. Moreover, we observe straight edges and a rectangular shape of the optimal decision boundary that can be attributed to the SAR(2) process equation.

When looking at the decision boundaries learned by the individual machine learning models, we gain a series of further insights: As expected, the LR model can only poorly approximate the strong nonlinearity of the SAR(2) decision boundary with a straight line resulting in the rather low performance in terms of predictive accuracy. Second, the two MLPs, the three tree based methods (CART, RAF, and GBT) and even kNN produce fairly good results with a decision boundary more or less resembling the optimum. In addition, we see signs of overfitting for the CART, i.e., very fine grained sub-areas throughout the plot. Lastly, the poor performance of NB (see previous paragraph) is also visually reflected in its decision boundary (see sub plot on the bottom right in figure 3).

Summary: From the base analysis we obtain the following key takeaways. First, MLPs and GBT are well suited for a broad variety of DGPs and come quite close to the ambition level. Second, in case of none or only weak nonlinearities, LR is a good choice. Third, NB, kNN, and CART are less suitable for time series prediction.

Lag analysis

The goal of the lag analysis is to assess how sensitive the performance of our machine learning models reacts to varying numbers of lagged values that are used as input features. This analysis is of particular importance as in general, we do not know the true, underlying

Figure 3: Decision boundaries obtained for SAR(2) process. The lighter shaded areas indicate a forecast value of "1", i.e., $\hat{s_t}$ is larger than or equal to s_{t-1}; the darker shaded areas a forecast value of "0". The x-axis denotes s_{t-1}, the y-axis s_{t-2}. The dots represent a subset of the actual observations that were used to fit the models with the color coding being similar to the one of the areas.

DGP that we aim to predict. In other words, we assess the effect of model mis-specifications, i.e., that a researcher selects too many or too few lags as input features.

Model performance: Table 4 depicts the average accuracy of our machine learning models

Model	Base case	Panel A: Number of lags 1	3	4	5	∅	Panel B: Relative change to base case 1	3	4	5	∅
LR	72.07%	69.85%	72.10%	72.10%	72.10%	71.54%	-3.08%	0.05%	0.05%	0.05%	-0.73%
MLP10	73.21%	71.33%	73.10%	72.99%	72.84%	72.56%	-2.57%	-0.15%	-0.30%	-0.50%	-0.88%
MLP100	**73.44%**	**71.66%**	**73.46%**	**73.38%**	**73.25%**	**72.94%**	-2.42%	0.02%	-0.09%	-0.27%	**-0.69%**
CART	69.61%	68.85%	69.19%	68.85%	68.61%	68.87%	**-1.09%**	-0.61%	-1.09%	-1.43%	-1.05%
RF	72.30%	69.03%	72.74%	72.67%	72.77%	71.80%	-4.52%	**0.61%**	**0.52%**	**0.65%**	**-0.69%**
GBT	72.95%	70.71%	72.99%	72.95%	72.90%	72.39%	-3.07%	0.06%	0.01%	-0.06%	-0.76%
kNN	70.49%	68.05%	70.29%	69.87%	69.32%	69.38%	-3.47%	-0.29%	-0.88%	-1.66%	-1.58%
NB	70.26%	69.14%	69.57%	69.62%	69.33%	69.42%	-1.59%	-0.98%	-0.91%	-1.32%	-1.20%
∅	**71.79%**	69.83%	**71.68%**	71.55%	71.39%	71.11%	-2.74%	**-0.16%**	-0.33%	-0.56%	-0.94%

Table 4: Panel A and B summarize the average accuracies of the machine learning models across all DGPs for different number of lags that are used as input features. Panel A contrasts the achieved accuracies with the base case in which two lagged values are used as input features. The column average at the bottom of Panel A depicts the average performance for a given number of lags across all models and DGPs. Panel B depicts the results of Panel A as relative change compared to the base case. The highest value of the respective column/row is highlighted in bold.

for a varying number of lagged values used as input features. When looking at the column average in Panel A, we see that the highest average accuracy of 71.79% is achieved in the base case in which the past two lagged values are used as features. This result is consistent with the process equations of the DGPs (see subsection 2.1.1) which all depend on the past two lagged values. We further notice that the inclusion of too few lagged values in the feature space, i.e., only one lag, leads to an average accuracy of 69.83% and hence has a significantly stronger negative effect than including more than two lags (average accuracy between 71.39% and 71.68%). Again, this is no surprise, as some of the DGPs, i.e., STAR(2), BL(2) and AR(2) heavily depend on the process realization at lag 2.

Looking at each of the machine learning models individually exhibits a more diverse picture. Again, the performance of all models decreases with one lag, however, we observe that this does not hold true for all models when using more than two lags as input features. Specifically, the RF benefits from feeding a larger number of lagged values leading to an increase in average accuracy from 72.30% in the base case up to 72.77% when the past five lags are used. We assume that the additional features lead to a stronger decorrelation of the trees and hence to a better prediction as result of reduced overfitting.

Summary: From the lag analysis we obtain the following key takeaways. First, using too few lags has a more negative effect on model performance than using too many. Second, random forests can even profit from additional lagged values.

Jump effect analysis

The goal of the jump effect analysis is twofold. First, we aim at assessing the impact of jumps on our machine learning models, i.e., we analyze the change in predictive accuracy for varying jump sizes. Second, we wish to compare different techniques to remove the jumps from the time series or to reduce their impact. In particular, we investigate the effect of using first differences (DIFF) as additional features as well as the effect of jump removal with the local outlier factor (LOF) method.

Model performance: Table 5 depicts the average accuracy of our machine learning models for jumps of size 0.1, 1 and 10 (see subsection 2.1.2). Looking at the column average in Panel A, we see a strong deterioration of the accuracies from 71.79% in the base case (no jumps) to 60.33% in case of large jumps. This result is well in line with the expectation that introducing greater discontinuities causes stronger mean shifts which in turn lead to a loss in predictive accuracy.

Model	**Base case**	**Panel A: Absolute jump size**			**∅**	**Panel B: Rel. change to base case**			**∅**
		0.1	**1**	**10**		**0.1**	**1**	**10**	
LR	72.07%	71.34%	66.28%	**66.22%**	**67.95%**	-1.00%	**-8.03%**	**-8.11%**	**-5.72%**
MLP10	73.21%	72.13%	66.61%	60.91%	66.55%	-1.46%	-9.01%	-16.79%	-9.09%
MLP100	**73.44%**	**72.32%**	**66.85%**	63.89%	67.69%	-1.53%	-8.97%	-13.01%	-7.84%
CART	69.61%	68.25%	62.52%	58.40%	63.06%	-1.95%	-10.19%	-16.11%	-9.41%
RF	72.30%	70.91%	64.88%	60.26%	65.35%	-1.92%	-10.26%	-16.65%	-9.61%
GBT	72.95%	71.50%	65.27%	60.33%	65.70%	-1.98%	-10.53%	-17.30%	-9.94%
kNN	70.49%	69.19%	63.94%	60.91%	64.68%	-1.85%	-9.30%	-13.60%	-8.25%
NB	70.26%	69.67%	59.85%	51.74%	60.42%	**-0.83%**	-14.82%	-26.37%	-14.01%
∅	71.79%	**70.67%**	64.52%	60.33%	65.17%	**-1.57%**	-10.12%	-15.96%	-9.22%

Table 5: Panel A and B summarize the average accuracies of the machine learning models across all DGPs for different jump sizes. Panel A contrasts the absolute values of the achieved accuracies with the base case. The column average at the bottom of Panel A depicts the average performance for a given jump size across all models and DGPs. Panel B depicts the results of Panel A as relative change compared to the base case. The highest value of the respective column/row is highlighted in bold.

When assessing the machine learning models on an individual level (Panel B of table 5), we obtain further insights. The LR proves to be the most robust machine learning model with a relative decline in predictive accuracy between -1.00% (small jumps) and -8.11% (large jumps). On average, the relative decline in accuracy of the LR amounts to -5.72% across all three jump sizes. In terms of absolute predictive accuracy (see Panel A), the LR even ranks first at a jump size of 10 rendering it the best model in presence of large jumps. Moreover, we find MLP100 (-7.84%), kNN (-8.25%) and MLP10 (-9.09%) to be slightly less affected than the tree-based methods CART (-9.41%), RF (-9.61%) and GBT (-9.94%). Finally, NB places last with an average relative decline of -14.01%.

Mitigation measures: In the second step, we aim at reducing the negative effect of jumps on our prediction problem. For the sake of comprehensiveness, we focus on the jump size of 1[10].

			Panel A: Measures				**Panel B: Rel. change to jump case**			
Model	**Base case**	**Jump case**	**DIFF**	**LOF**	**DIFF LOF**	∅	**DIFF**	**LOF**	**DIFF LOF**	∅
LR	72.07%	66.28%	68.24%	69.49%	70.00%	69.24%	2.96%	4.85%	5.61%	4.48%
MLP10	73.21%	66.61%	68.60%	69.79%	70.28%	69.56%	2.98%	4.77%	5.51%	4.42%
MLP100	**73.44%**	**66.85%**	**68.65%**	**69.97%**	**70.48%**	**69.70%**	2.69%	4.65%	5.42%	4.25%
CART	69.61%	62.52%	63.89%	65.20%	65.55%	64.88%	2.20%	4.29%	4.86%	3.78%
RF	72.30%	64.88%	66.93%	68.05%	68.86%	67.95%	3.16%	4.88%	6.13%	4.72%
GBT	72.95%	65.27%	67.33%	68.81%	69.50%	68.55%	3.17%	5.44%	6.49%	5.03%
kNN	70.49%	63.94%	65.39%	66.17%	66.90%	66.15%	2.28%	3.49%	4.63%	3.47%
NB	70.26%	59.85%	66.23%	67.51%	69.35%	67.70%	**10.66%**	**12.81%**	**15.87%**	**13.11%**
∅	**71.79%**	64.52%	66.91%	68.12%	**68.86%**	67.97%	3.70%	5.58%	**6.73%**	5.41%

Table 6: Panel A and B summarize the average accuracies of the machine learning models for the jump size of 1 across all DGPs for different mitigation measures, i.e., use of first differences as additional features (DIFF), jump removal with the local outlier factor method (LOF), and the combination of the two (DIFF LOF). Panel A contrasts the achieved accuracies with the jump case (no mitigation measures) and the base case (no jumps). The column average at the bottom of Panel A depicts the average performance for a given mitigation technique across all machine learning models and DGPs. Panel B depicts the results of Panel A as relative change compared to the jump case. The highest value of the respective column/row is highlighted in bold.

[10]The results for smaller and larger jumps sizes are similar to those at jump size 1 however the effects are smaller at jump size 0.1 and larger at jump size 10 respectively.

Table 6 depicts the impact of different mitigation measures and contrasts them with the jump case (jumps of size 1 without mitigation measures) as well as the base case (no jumps). Specifically, we assess the effect of first differences as additional features (DIFF), the impact of jump removal with LOF, and the combination of the two (DIFF LOF). Looking at the column average in Panel A, we find a positive effect for all three techniques leading to an increase in predictive accuracy from 64.52% (jump case) up to 68.86% (+6.73%) when applying the combination of DIFF and LOF.

Looking at the measures individually, we find LOF to have the strongest individual effect of +5.58% compared to the jump case. During the preprocessing of the time series with LOF, a large fraction of the jumps and hence mean shifts is removed. The resulting time series therefore more closely resembles the one prior to jumps (base case) and the accuracy improves. Also, the introduction of first differences (DIFF) as additional features helps the machine learning models to cope with the mean shift. Even though the jumps are still present in the time series, they now only affect the training samples that capture the time span during which the jumps occurred and hence only a very small fraction of the observations used to fit the machine learning models. As result, the accuracy increases by +3.70% relative to the jump case.

When looking at the machine learning models individually, we find that all models benefit from the mitigation measures. The ranking of the measure effects is consistent with the averaged ranking outlined above, i.e., the combination of DIFF and LOF yields better effects than LOF, which in turn yields better effects than DIFF alone across all models. NB accounts for the strongest relative improvement (+15.87% for DIFF LOF), however NB had also shown the strongest decline prior to mitigation measures. The opposite applies to LR which turned out to be most robust to jumps and hence shows the lowest improvement when applying mitigation measures.

Summary: From the jump effect analysis we obtain the following key takeaways. First, jumps significantly worsen the performance across all machine learning models with the effect increasing with higher jump sizes. Second, LR is fairly robust to jumps and hence a good choice in absence of mitigation measures. Third, both the use of first differences as additional features (DIFF) as well as jump removal with local outlier factor (LOF) significantly improve performance in light of jumps. The best results can be achieved by combining both techniques.

Noise analysis

The noise analysis serves two purposes. First, we aim at assessing the impact of noise on our machine learning models by analyzing the change in predictive accuracy when distorting the DGP with noise of low, medium and high degree. Second, we evaluate and compare different techniques to reduce the noise. In particular, we examine the effect of using first differences (DIFF), moving average (MA) as well as the combination of the two (DIFF MA) as additional features.

Model performance: Table 7 depicts the average accuracy of our machine learning models when polluting the DGP with varying degrees of noise (see subsection 2.1.3). From the column average in Panel A, we observe a decline in accuracy from 71.79% in the base case up until 56.72% in presence of high noise. The underlying reason is as follows: As the time series is superposed with noise in form of a random walk process, the time series becomes more and more non-stationary (due to the mean drift) and hence harder to predict. This effect becomes stronger with increasing variance of the random walk process which corresponds to higher degrees of noise.

Looking at the row sum of Panel B of table 7, we observe that the negative effect of noise strongly varies between the individual machine learning models. As in the jump effect analysis, the LR turns out to be the most robust model with a relative decline in accuracy of -12.02% compared to the base case and first rank in terms of absolute predictive accuracy across all degrees of noise. Furthermore, we observe that the tree based models, i.e., CART (-19.47%), RF (-18.88%), and GBT (-19.27%), react more sensitive to noise compared to the MLP100 (-14.64%), the MLP10 (-15.73%), and kNN (-17.24%). We assume that the tree-based methods (DT, RF, and GBT) are more affected by the mean drift as it significantly changes the threshold values at which splits occur in the nodes of the trees. Finally, NB ranks last with an average relative decline in accuracy of -25.03%.

Mitigation measures: In step 2, we aim at mitigating the negative effect of noise on our prediction problem. Again, we focus on one noise level, i.e., medium noise, to keep the

Model	Base case	Panel A: Noise level low	medium	high	∅	Panel B: Rel. change to base case low	medium	high	∅
LR	72.07%	**65.57%**	**63.28%**	**61.36%**	**63.40%**	**-9.02%**	**-12.19%**	**-14.86%**	**-12.02%**
MLP10	73.21%	65.11%	61.34%	58.63%	61.69%	-11.05%	-16.22%	-19.91%	-15.73%
MLP100	**73.44%**	65.38%	62.52%	60.17%	62.69%	-10.98%	-14.87%	-18.08%	-14.64%
CART	69.61%	59.11%	55.33%	53.72%	56.05%	-15.09%	-20.51%	-22.83%	-19.47%
RF	72.30%	61.66%	58.09%	56.20%	58.65%	-14.72%	-19.66%	-22.26%	-18.88%
GBT	72.95%	61.90%	58.35%	56.43%	58.89%	-15.15%	-20.02%	-22.65%	-19.27%
kNN	70.49%	61.05%	57.89%	56.08%	58.34%	-13.39%	-17.88%	-20.44%	-17.24%
NB	70.26%	54.89%	51.98%	51.16%	52.67%	-21.88%	-26.02%	-27.19%	-25.03%
∅	71.79%	**61.83%**	58.60%	56.72%	59.05%	**-13.87%**	-18.38%	-20.99%	-17.75%

Table 7: Panel A and B summarize the average accuracies of the machine learning models across all DGPs for different degrees of noise. Panel A contrasts the absolute values of the achieved accuracies with the base case. The column average at the bottom of Panel A depicts the average performance for a given noise level (low/medium/high) across all models and DGPs. Panel B depicts the results of Panel A as relative change compared to the base case. The highest value of the respective column/row is highlighted in bold.

analysis clear and comprehensible[11].

Table 8 summarizes the impact of our three mitigation measures and contrasts them with the medium noise case as well as the base case (no noise). Specifically, we examine the effect of first differences (DIFF), moving averages (MA) and the combination of the two (DIFF MA) as additional features. The column average in Panel A indicates a positive effect for all three techniques leading to accuracies from 59.30% (MA) up to 62.89% (DIFF) compared to 58.60% without mitigation measures. We find DIFF to be the single best technique increasing accuracy by +7.32% relative to the medium noise case. First differencing (DIFF) makes the time series fully stationary, facilitating the machine learning models' ability to identify meaningful structure. Also, using moving averages as additional features has a positive effect, however with +1.20% at significantly smaller scale compared to DIFF. Finally, the combination of DIFF and MA leads to an improvement of +7.03% on average - slightly lower than DIFF alone. However, when looking at the effects on the machine learning models individually, we observe that in some cases, i.e., LR, MLP10 and MLP100, DIFF MA yields slightly better results than DIFF alone. It is hence worth experimenting with this

[11] The results are similar for low and high noise however at a lower/higher scale.

			Panel A: Measures			Panel B: Relative change		
Model	**Base case**	**Medium noise**	**DIFF**	**MA**	**DIFF MA**	**DIFF**	**MA**	**DIFF MA**
LR	72.07%	**63.28%**	**64.97%**	**65.38%**	**65.39%**	2.66%	**3.31%**	3.33%
MLP10	73.21%	61.34%	64.74%	62.68%	64.81%	5.55%	2.20%	5.66%
MLP100	**73.44%**	62.52%	64.93%	64.05%	64.99%	3.84%	2.44%	3.95%
CART	69.61%	55.33%	59.72%	55.26%	59.69%	7.92%	-0.14%	7.88%
RF	72.30%	58.09%	62.38%	58.89%	62.31%	7.39%	1.38%	7.26%
GBT	72.95%	58.35%	62.51%	58.94%	62.43%	7.14%	1.02%	7.00%
kNN	70.49%	57.89%	60.49%	58.04%	60.24%	4.49%	0.26%	4.06%
NB	70.26%	51.98%	63.38%	51.18%	61.86%	**21.93%**	-1.54%	**19.01%**
∅	71.79%	58.60%	**62.89%**	59.30%	62.71%	**7.32%**	1.20%	7.03%

Table 8: Panel A and B summarize the average accuracies of the machine learning models across all DGPs at medium noise level for different mitigation measures, i.e., use of first differences (DIFF), moving averages (MA) and the combination of the two (DIFF MA) as additional features. Panel A contrasts the achieved accuracies with the medium noise case (no mitigation measures) and the base case (no noise). The column average at the bottom of Panel A depicts the average performance for a given noise reduction technique across all machine learning models and DGPs. Panel B depicts the results of Panel A as relative change compared to the medium noise case. The highest value of the respective column/row is highlighted in bold.

combination when working with noisy time series data.

Summary: From the noise analysis we obtain the following key takeaways. First, the presence of noise in time series data significantly worsens the performance across all machine learning models. Clearly, the effect increases with higher levels of noise. Second, as in the jump case, LR proves to be the most robust model for noisy time series data, ranking first in terms of performance - even after noise mitigation measures. Third, the use of first differences (DIFF), moving averages (MA), and the combination of both (DIFF MA) as additional features all lead to improvements in predictive accuracy. Depending on the applied machine learning model, DIFF leads to slightly better results than the combination DIFF MA.

Conclusion

In this paper, we perform a comprehensive simulation study contrasting the predictive performance of eight different machine learning models on a great variety of linear and nonlinear time series. Specifically, we conduct 29 Monte Carlo experiments to thoroughly analyze the performance of each model as well as the impact of jumps as well as varying degrees of noise. We make the following key findings.

First, we find machine learning models to achieve solid performance on unknown underlying DGPs, compared to the ambition level set by optimal forecasts. In absence of noise (base case), the results achieved with the machine learning models almost resemble those of the optimal forecast. Model-wise, MLPs and GBT provide the best results for both, linear and nonlinear DGPs. For processes with no or only small nonlinearities, the LR presents a good alternative, especially when considering its low computational cost. We find NB and single decision trees to deliver worse performance and hence recommend the aforementioned techniques for time series prediction tasks.

Second, it is better to include too many lagged values in the feature space than to include too few. We find most machine learning models to be fairly robust to exceeding the number of required lags suggested by the process equation of the DGP. In case of the RF, the inclusion of additional features even increases the predictive accuracy. We recommend to start with one lag and to gradually increase the number of lags monitoring the performance on a hold out set or with cross validation.

Third, we find jumps to have a very strong negative effect on predictive accuracy with LR being the most robust machine learning model. To mitigate the negative effects, both adding first differences to the features space (DIFF) as well as removal of jumps based on the LOF algoirthm have shown good results. We recommend the combination of both techniques.

Fourth, polluting the time series with noise has the most detrimental effect on predictive accuracy across all machine learning models. Again, we find that the LR is the most robust machine learning model in the presence of noise. Moreover, additional mitigation measures, such as the inclusion of first differences (DIFF) and moving averages (MA) in the feature space yield improved results.

References

Ahmed, N. K., Atiya, A. F., Gayar, N. E., El-Shishiny, H., 2010. An empirical comparison of machine learning models for time series forecasting. Econometric Reviews 29 (5-6), 594–621.

Alon, I., Qi, M., Sadowski, R. J., 2001. Forecasting aggregate retail sales: A comparison of artificial neural networks and traditional methods. Journal of Retailing and Consumer Services 8 (3), 147–156.

Bergmeir, C., Benítez, J. M., 2012. On the use of cross-validation for time series predictor evaluation. Information Sciences 191, 192–213.

Bergmeir, C., Costantini, M., Benítez, J. M., 2014. On the usefulness of cross-validation for directional forecast evaluation. Computational Statistics & Data Analysis 76, 132–143.

Bergmeir, C., Hyndman, R. J., Koo, B., 2015. A note on the validity of cross-validation for evaluating time series prediction. Monash University, Department of Econometrics and Business Statistics, Tech. Rep.

Bishop, C. M., 2007. Pattern Recognition and Machine Learning (Information Science and Statistics), 1st edn. 2006. corr. 2nd printing edn. Springer, New York.

Bollerslev, T., 1986. Generalized autoregressive conditional heteroskedasticity. Journal of Econometrics 31 (3), 307–327.

Bontempi, G., Taieb, S. B., Le Borgne, Y. A., 2013. Machine learning strategies for time series forecasting. Business Intelligence, 62–77.

Box, G., Jenkins, G., 1970. Time series analysis, forecasting and control. Holden-Day, San Francisco.

Breiman, L., 2001. Random forests. Machine Learning 45 (1), 5–32.

Breunig, M. M., Kriegel, H. P., Ng, R. T., Sander, J., 2000. LOF: Identifying density-based local outliers. ACM sigmoid record 29 (2), 93–104.

Callen, J. L., Kwan, C. C., Yip, P. C., Yuan, Y., 1996. Neural network forecasting of quarterly accounting earnings. International Journal of Forecasting 12 (4), 475–482.

Chakraborty, K., Mehrotra, K., Mohan, C. K., & Ranka, S., 1992. Forecasting the behavior of multivariate time series using neural networks. Neural networks 5 (6), 961–970.

Chan, K. S., Tong, H., 1986. On estimating thresholds in autoregressive models. Journal of Time Series Analysis 7 (3), 179–190.

Connor, J. T., Martin, R. D., Atlas, L. E., 1994. Recurrent neural networks and robust time series prediction. IEEE transactions on neural networks 5 (2), 240–254.

Cox, J. C., Ross, S. A., 1976. The valuation of options for alternative stochastic processes. Journal of Financial Economics 3 (1-2), 145–166.

Dreiseitl, S., Ohno-Machado, L., 2002. Logistic regression and artificial neural network classification models: a methodology review. Journal of Biomedical Informatics 35 (5), 352–359.

Durante, D., Scarpa, B., Dunson, D. B., 2014. Locally adaptive factor processes for multivariate time series. Journal of Machine Learning Research 15 (1), 1493–1522.

Engle, R. F., 1982. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. Econometrica 50 (4), 987.

Friedman, J. H., 2002. Stochastic gradient boosting. Computational Statistics & Data Analysis 38 (4), 367–378.

Gooijer, J. G. d., Hyndman, R. J., 2006. 25 years of time series forecasting. International Journal of Forecasting 22 (3), 443–473.

Granger, C., Anderson, A. P., 1978. An introduction to bilinear time series models. Vandenhoeck & Ruprecht, Göttingen.

Hand, D. J., Yu, K., 2001. Idiot's Bayes? Not so stupid after all? International Statistical Review 69 (3), 385–398.

Hastie, T., Tibshirani, R., Friedman, J., 2008. The elements of statistical learning: Data mining, inference, and prediction, 2nd Edition. Springer Series in Statistics, New York.

Hill, T., O'Connor, M., Remus, W., 1996. Neural network models for time series forecasts. Management Science 42 (7), 1082–1092.

Ho, T. K., 1995. Random decision forests. In: Proceedings of the third International Conference on Document Analysis and Recognition. Vol. 1. IEEE, pp. 278–282.

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the International Conference on Machine Learning. pp. 448–456.

Jondeau, E., Lahaye, J., Rockinger, M., 2015. Estimating the price impact of trades in a high-frequency microstructure model with jumps. Journal of Banking & Finance 61, S205–S224.

Kaastraa, I., Boyd, M., 1996. Designing a neural network for forecasting financial and economic time series. Neurocomputing 10 (3), 215–236.

Khaleghi, A., Ryabko, D., Mari, J., Preux, P., 2016. Consistent algorithms for clustering time series. Journal of Machine Learning Research 17 (3), 1–32.

Kim, K., 2003. Financial time series forecasting using support vector machines. Neurocomputing 55 (1-2), 307–319.

Kohavi, R., et al., 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceeding of the International Joint Conference on Artificial Intelligence. Vol. 14. Stanford, CA, pp. 1137–1145.

Krollner, B., Vanstone, B., Finnie, G., 2010. Financial time series forecasting with machine learning techniques: A survey. European Symposium on Artificial Neural Networks: Computational and Machine Learning.

Lewis, D. D., 1998. Naive (bayes) at forty: The independence assumption in information retrieval. In: Proceedings of the European conference on machine learning. Springer, pp. 4–15.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perror, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.

Qian, G., Sural, S., Gu, Y., Pramanik, S., 2004. Similarity between euclidean and cosine angle distance for nearest neighbor queries. In: Proceedings of the 2004 ACM symposium on Applied computing. ACM, pp. 1232–1237.

Raschka, S., 2014. Naive bayes and text classification I - Introduction and Theory. arXiv preprint arXiv:1410.5329.

Raschka, S., 2015. Python Machine Learning. Packt Publishing, Birmingham.

Robinson, J. W., Hartemink, A. J., 2010. Learning non-stationary dynamic Bayesian networks. Journal of Machine Learning Research 11 (Dec), 3647–3680.

Ross, S. M., 2007. Stochastic processes, 2nd Edition. Wiley series in probability and statistics. Wiley, New York.

Schapire, R. E., 1990. The strength of weak learnability. Machine learning 5 (2), 197–227.

Taylor, S. J., 1987. Forecasting the volatility of currency exchange rates. International Journal of Forecasting 3 (1), 159–170.

Terasvirta, T., Anderson, H. M., 1992. Characterizing nonlinearities in business cycles using smooth transition autoregressive models. Journal of Applied Econometrics 7 (S1), S119–S136.

Terasvirta, T., Tjostheim, D., Granger, C. W., 2010. Modelling nonlinear economic time series. OUP Catalogue.

Tong, H., 2012. Threshold models in non-linear time series analysis. Springer, New York.

Tong, H., Chan, K. S., Cox, D. R., Cutler, C., Guégan, D., Jensen, J. L., Johansen, S., Lawrance, A. J., LeBaron, B., Ozaki, T., Nychka, D. W., Ellner, S., Bailey, B. A., Gallant, A. R., Smith, L. R., Smith, R. L., Wolff, R. C. L., 1995. A personal overview of non-linear time series analysis from a chaos perspective (with discussion and rejoinder). Scandinavian Journal of Statistics 22 (4), 399–445.

Tong, H., Lim, K. S., 1980. Threshold autoregression, limit cycles and cyclical data. Journal of the Royal Statistical Society. Series B 42 (3), 245–292.

Xiao, Y., Xiao, J., Wang, S., 2012. A hybrid model for time series forecasting. Human Systems Management 31 (2), 133–143.

Yule, G. U., 1927. On a method of investigating periodicities in disturbed series, with special reference to Wolfer's sunspot numbers. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 226 (636-646), 267–298.

Zhang, G., Patuwo, B. E., Hu, M. Y., 1998. Forecasting with artificial neural networks: The state of the art. International Journal of Forecasting 14 (1), 35–62.

Zhang, G. P., 2003. Time series forecasting using a hybrid ARIMA and neural network model. Neurocomputing 50, 159–175.

Zhang, G. P., Patuwo, B. E., Hu, M. Y., 2001. A simulation study of artificial neural networks for nonlinear time-series forecasting. Computers & Operations Research 28 (4), 381–396.

Zhang, G. P., Qi, M., 2005. Neural network forecasting for seasonal and trend time series. European Journal of Operational Research 160 (2), 501–514.